

AD-A172 516

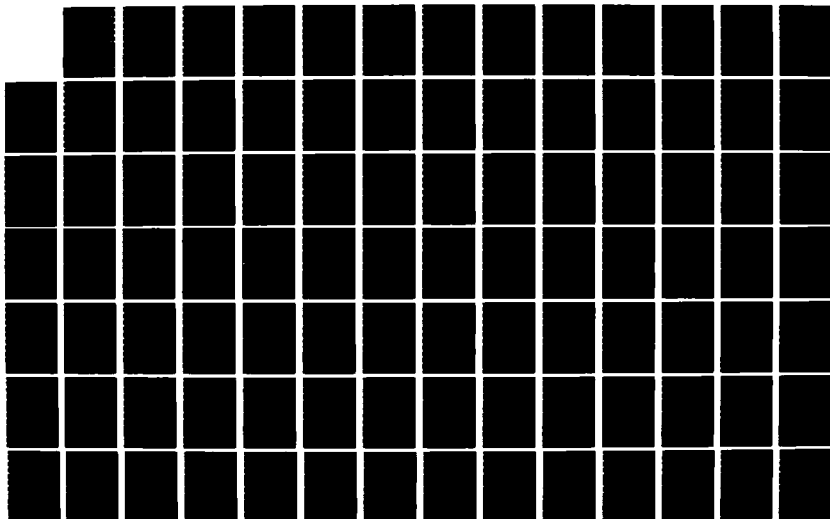
A NON-COGNITIVE FORMAL APPROACH TO KNOWLEDGE  
REPRESENTATION IN ARTIFICIAL (U) AIR FORCE INST OF  
TECH WRIGHT-PATTERSON AFB OH SCHOOL OF ENGI  
J A MCMAHAMA JUN 86 AFIT/DS/ENG/86J-1

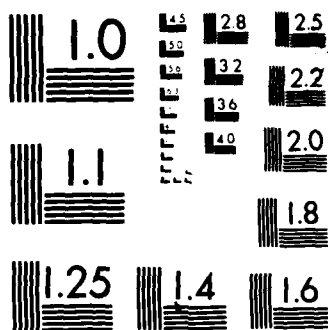
1/4

UNCLASSIFIED

F/G 9/4

NL

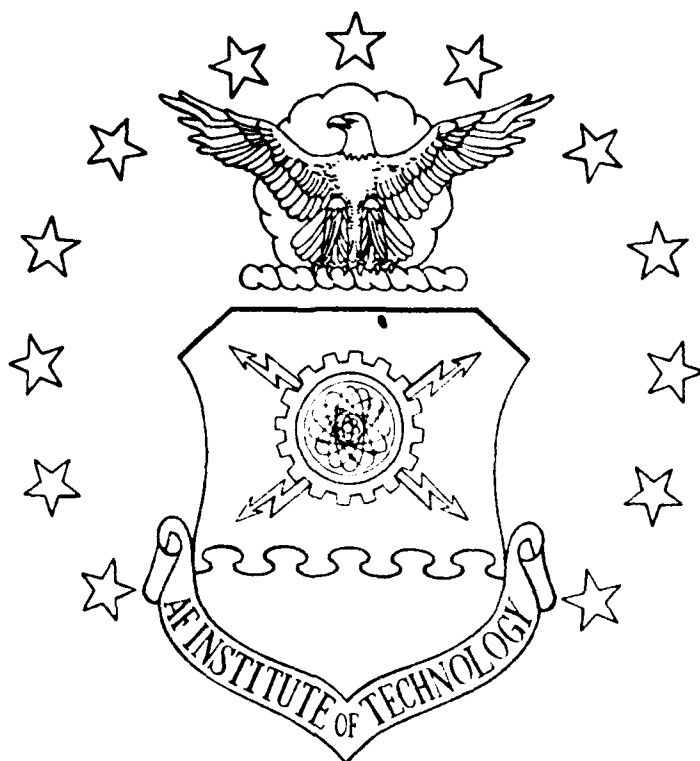




MICROCOPY RESOLUTION TEST CHART  
NATIONAL BUREAU OF STANDARDS-1963-A

AD-A172 516

DTIC FILE COPY



A NON-COGNITIVE FORMAL APPROACH TO  
KNOWLEDGE REPRESENTATION IN  
ARTIFICIAL INTELLIGENCE

DISSERTATION

Jim A. McMannama  
Major, USAF

AFIT/DS/ENG/86J-1

DEPARTMENT OF THE AIR FORCE  
AIR UNIVERSITY

**AIR FORCE INSTITUTE OF TECHNOLOGY**

Wright-Patterson Air Force Base, Ohio

86 10 01 205

0

A NON-COGNITIVE FORMAL APPROACH TO  
KNOWLEDGE REPRESENTATION IN  
ARTIFICIAL INTELLIGENCE

DISSERTATION

Jim A. McMannama  
Major, USAF

AFIT/DS/ENG/86J-1

Approved for public release; distribution unlimited



AFIT/DS/ENG/86J-1

A NON-COGNITIVE FORMAL APPROACH TO KNOWLEDGE REPRESENTATION  
IN ARTIFICIAL INTELLIGENCE

DISSERTATION

Presented to the Faculty of the School of Engineering  
of the Air Force Institute of Technology

Air University

In Partial Fulfillment of the  
Requirements for the Degree of  
Doctor of Philosophy

Jim A. McMannama, B.S., M.S.

Major, USAF

June 1986

Accession For	
NTIS GRA&I	51
DTIC TAB	7
Unannounced	
Justification	
By	
Distribution	
Availability Codes	
Avail and/or	
Restrictions	
AI	

Approved for public release; distribution unlimited

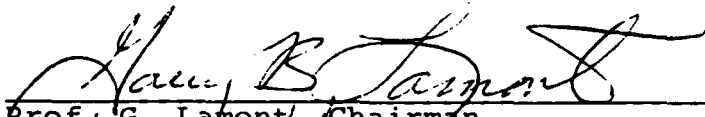


A NON-COGNITIVE FORMAL APPROACH TO KNOWLEDGE REPRESENTATION  
IN ARTIFICIAL INTELLIGENCE


Jim A. McMannama, B.S., M.S.

Major, USAF


Approved:

  
Prof. G. Lamont, Chairman 3 June 1986


  
Prof. M. Kabrisky 16 May 86.

  
Prof. H. Potoczny June 2, 1986.

  
Assist. Prof. Major S. Cross 2 June 86

  
Prof. F. Brown 3 JUNE 1986

Accepted:

  
Dean, School of Engineering

## Preface

The purpose of this study was to add formality to knowledge representation in the Artificial Intelligence (AI) field. Because knowledge representations could be considered analogous to abstract data types, I felt that there must be a variety of formal analysis techniques already developed by computer scientists that could be applied directly to my formalization task. In pursuit of this general concept, I found that formal language researchers had developed such a set of techniques. By showing the usefulness of these existing techniques in this dissertation, I feel that AI researchers can now be prevented from "re-inventing the wheel" in their search for formal techniques to use in analyzing knowledge representations.

Because of the numerous articles I reviewed in search of the techniques, I chose only to cite a representative sample within the body of the dissertation. However, my views on the dissertation subject could not help but be swayed by all the articles I did review. Therefore, to provide you with a better understanding of my position, I placed in Appendices A and B my extensive bibliography on formal language theory and AI knowledge representation. I make no claims as to the completeness of the lists.

Literature articles were not the only help I received. I wish to thank my entire advisory committee for the encouragement they provided me, especially when a different approach I had taken failed to produce results. I am particularly indebted to my committee chairman, Prof. Gary B. Lamont, for the ideas that we generated during our weekly brainstorming sessions and for his understanding and guidance during some of the personal tribulations that I experienced during my AFIT tour. Additionally, I wish to thank my office mates, Capt Edward McCall, Capt Timothy Kearns and Capt Charles Matson for the informative interchanges of opinions on my topic.

Because of the extensive literature search that I performed, I wish also to acknowledge the outstanding efforts of the AFIT engineering library staff for their rapid responses to my requests for books and articles from other libraries. Also, many thanks go out to my wife Karyle and my daughters Wendi and Tiffany for the many sacrifices that they made so that I would have sufficient study time. Finally, I want to thank my parents Loren (deceased) and Hilma for the financial and emotional support they provided so that I could obtain my initial college education without which this dissertation could never have been undertaken.

Jim A. McMannama

## Table of Contents

	Page
Preface.....	iii
List of Figures.....	viii
List of Tables.....	ix
List of Definition Numbers.....	x
List of Theorem Numbers.....	xi
List of Corollary Numbers.....	xiii
List of Conjecture Numbers.....	xiv
List of Proposition Numbers.....	xv
Abstract.....	xvi
I. Introduction.....	1
Background.....	1
Intelligent Computers.....	2
Artificial Intelligence Definition.....	5
Formal Language Theory.....	7
Definitions and Characteristics.....	7
Languages and Grammars.....	9
Language Problems.....	12
Relationship of AI and Languages.....	14
Individual Knowledge Representations.....	16
Predicate Calculus.....	17
Production Rules.....	19
Semantic Networks.....	23
Frames.....	27
Scripts.....	28
Conceptual Dependency.....	30
Combined Representations.....	33
Knowledge Representation Relationships...	35
The Problem.....	37
Problem Background.....	38
Problem Statement.....	39

	Page
II. Dissertation Approach.....	42
General Concept.....	42
Supporting Rationale.....	42
Counterpoints.....	44
Approach.....	48
Scope.....	48
Overview.....	51
Reader's Guidance.....	54
III. Language Space.....	58
Supporting Definitions and Theorems.....	59
Language-Space Size.....	65
IV. Representation Definitions.....	73
Definition of Production Rules.....	73
Definition of Semantic Networks.....	79
Related Definitions.....	83
V. Production-Rule Language and Characteristics.....	85
Production-Rule Language.....	85
Case I: Facts are Words.....	86
Case II: Facts are Symbols.....	92
Production-Rule Characteristics.....	95
PA Characteristics.....	95
Category 1 Class.....	99
VI. Semantic-Network Language and Characteristics....	109
Semantic-Network Language.....	109
Case I: Word Approach.....	109
Case II: Symbol Approach.....	127
Semantic-Network Characteristics.....	132
SA Characteristics.....	132
Category 1 Class.....	138
VII. Comparison of Rules and Networks.....	148
Equivalences.....	148
Transformations.....	165
VIII. Hierarchical Language Results.....	178
Hierarchical Language Expansion.....	178
Hierarchical Equivalences.....	194
Hierarchical Transformations.....	202

	Page
IX. Conclusions and Discussion.....	214
Conclusions.....	214
Discussion.....	223
X. Recommendations and Closing.....	227
Recommendations.....	227
Closing.....	233
Appendix A: References on Formal Language Theory.....	236
Appendix B: References on Knowledge Represenataion.....	262
Appendix C: Invalid Nonenumerable Set.....	271
Appendix D: Grammar Space.....	273
Bibliography.....	283
Vita.....	292

## List of Figures

Figure	Page
1. Semantic Network Bag Model.....	4
2. Neural Network Word-Model.....	5
3. Predicate Calculus Example.....	19
4. Expert System Functional Diagram.....	21
5. Production Rule Example.....	22
6. Quillian's Semantic Memory.....	24
7. Semantic Network Example.....	26
8. Frame Representation Example.....	29
9. Script Example.....	31
10. Conceptual Dependency Example.....	32
11. Dissertation Flow Diagram.....	56
12. Inclusion Hierarchy.....	76
13. PR Word Matrix.....	87
14. PF Word Matrix.....	98
15. $Y^+$ Acceptor.....	115
16. SN Acceptor.....	117
17. Equivalent SN Acceptor.....	119
18. Node-Label and Relation Sets.....	137



## List of Tables

Table	Page
I. Cognitive Models.....	3
II. Categories of Antecedents of Production Rules.....	74
III. Categories of Consequents of Production Rules.....	75
IV. Categories of Nodes of Semantic Networks.....	80
V. Categories of Individual Links of Semantic Networks.....	81
VI. Categories of Link Combinations of Semantic Networks.....	81

# List of Definition Numbers

Definition Number	Page
1.01.....	6
1.02.....	7
1.03.....	8
1.04.....	8
1.05.....	9
1.06.....	9
3.01.....	59
3.02.....	60
3.03.....	60
3.04.....	60
3.05.....	60
3.06.....	60
3.07.....	60
3.08.....	60
3.09.....	61
3.10.....	61
3.11.....	61
3.12.....	61
3.13.....	61
3.14.....	61
3.15.....	62
3.16.....	62
3.17.....	63
3.18.....	63
3.19.....	63
3.20.....	64
3.21.....	64
3.22.....	64
3.24.....	64
4.01.....	77
4.02.....	82
4.03.....	83
4.04.....	84
5.01.....	93
6.01.....	128
7.01.....	149

# List of Theorem Numbers

Theorem Number	Page
1.01.....	9
3.01.....	61
3.02.....	61
3.03.....	61
3.04.....	62
3.05.....	62
3.06.....	62
3.07.....	62
3.08.....	62
3.09.....	62
3.10.....	62
3.11.....	64
3.12.....	65
3.13.....	65
3.14.....	66
3.15.....	67
3.16.....	67
3.17.....	68
3.18.....	68
3.19.....	69
3.20.....	70
5.01.....	86
5.02.....	87
5.03.....	89
5.04.....	90
5.05.....	90
5.06.....	92
5.07.....	93
5.08.....	94
5.09.....	96
5.10.....	96
5.11.....	97
5.12.....	98
5.13.....	100
5.14.....	102
5.15.....	103
5.16.....	105
5.17.....	106
6.01.....	110
6.02.....	111
6.03.....	112
6.04.....	113
6.05.....	114
6.06.....	123

Theorem Number	Page
6.07.....	128
6.08.....	129
6.09.....	130
6.10.....	131
6.11.....	132
6.12.....	133
6.13.....	134
6.14.....	135
6.15.....	138
6.16.....	141
6.17.....	143
6.18.....	145
6.19.....	146
7.01.....	149
7.02.....	155
7.03.....	157
7.04.....	158
7.05.....	160
7.06.....	164
7.07.....	165
7.08.....	172
8.01.....	179

List of Corollary Numbers

Corollary Number	Page
3.01.....	66
3.02.....	68

List of Conjecture Numbers

Conjecture Number	Page
8.01.....	183
8.02.....	183
8.03.....	196

List of Proposition Numbers

Proposition Number	Page
8.01.....	184
8.02.....	188
8.03.....	191
8.04.....	192
8.05.....	196
8.06.....	199
8.07.....	203
8.08.....	205

## Abstract

With the entry of Artificial Intelligence (AI) into real-time applications (e.g. Defense Department's Pilot's Associate Program), a rigorous analysis of AI expert systems is required in order to validate them for operational use. This analysis effort should include the formal mathematical analysis of the individual computer program elements contained within these expert systems. Included in this set of elements are the various knowledge representation schemes used for the knowledge-base.

To satisfy this requirement for analysis of knowledge representations, the techniques of formal language theory are used. A combination of theorems, proofs and problem-solving techniques from formal language theory are employed, either directly or in a modified form, to analyze language equivalents of the more commonly used AI knowledge representations of production rules (excluding working memory or situation data) and semantic networks.

This analysis reveals specific characteristics about these two representations such as their formal language-type, representation set-size, closure properties, special-case equivalences, special-case transformations and special representation-variations. Because of the complexity of formal language theory, many examples are provided to help clarify these characteristics.



By studying several of these characteristics in combination, two major results are identified that affect the use of these two representations. The first result stems from the characteristic that there is a countably infinite number of possible knowledge-languages (the sets of antecedent/consequent words, node-label words and relation-words) that can be contained in either a production-rule or a semantic-network structure. Using this characteristic, no single support-tool or automatic-programming tool can ever be constructed that will be able to handle all possible production-rule or semantic-network variations.

The second result stems from the characteristic that while the set of all finite production-rule languages and the set of all finite semantic-network languages are not equivalent, there are unique subsets of these languages that can be transformed into each other by a single transformation. Using this transformation, the entire set of finite production-rule languages is shown to be able to be stored in and retrieved from some of the finite semantic-network languages. In effect, the semantic-network structure is a viable candidate for a centralized database of knowledge.

# A NON-COGNITIVE FORMAL APPROACH TO KNOWLEDGE REPRESENTATION IN ARTIFICIAL INTELLIGENCE

## I. Introduction

### Background

One of the oldest subjects of interest to philosophers, linguists, psychologists, etc., is human intelligence. In the past, they have attempted to define intelligence as various collections of observed human behaviors. Based on the characteristics of these behaviors, they proposed theories and models that could generate the same "intelligent" behavior patterns. For example, the behavioral psychologists' stimulus-response pair model is used extensively in their research (1:4).

Because these older models of human intelligence were based on example observations, many questions were raised regarding the ability of the models to accurately simulate all human beings. As advances in medical technology provided increased knowledge about the human brain, these questions of accuracy were handled by expanding the older models to include known brain operations. These more advanced models are known as the neural networks (2). However, the brain operations that are being simulated are still based on nondestructive observations. Therefore,

these more advanced models are still relying on example observations and are now at a lower modeling level than was previously possible.

Intelligent Computers. As these advances in knowledge about the brain were being made, another technology was being developed which would significantly impact all of the models of human intelligence. That was the development of the digital computer. Various sequential processes that were thought to generate intelligent behavior patterns could also be modeled via computer programs. For example, Newell's and Simon's Logic Theorist model was implemented in a computer program and studied via that medium (1:3). Of course, the use of computers assumes that human intelligence is a logical process.

At the same time these more advanced psychological models were being studied, a specialty area was developing which studied computer programs that exhibited "intelligent behavior": Artificial Intelligence (AI). The psychological models and the computer program models appeared to come together when Quillian's computer program model called a semantic network was classified as a psychological model of human memory (1:8).

Quillian's tie between psychological models and computer programs led to the generation of the term "Cognitive Science." Cognitive Science involves the study of computer models of human thinking (1:4). Examples of such models that have been studied to date are given in Table I.

Table I. Cognitive Models (1:Section XI)

Problem Solvers: General Problem Solver.

Opportunistic Problem Solver.

Memory Models: Semantic Network

Human Associative Memory.

MEMOD.

ACT.

Learning: Elementary Perceiver and Memorizer (EPAM).

Belief Systems: PARRY.

An outstanding characteristic of these cognitive models is that they are developed from a macro-level point of view. That is, cognitive models use high level abstract definitions of the internal brain structure and perform functions based on that definition. For example, Figure 1 shows a semantic network representation of the relationship of Golf\_Bag to Bags and Bags to Containers. The semantic-network model uses an abstraction of nodes and links to model how the brain stores information. How an individual symbol, node or link is actually stored within memory is not of direct importance to the model definition. Only the node/link string names and the existence of link ties are of concern (see the subsection on semantic networks in Section I for more details on this model).

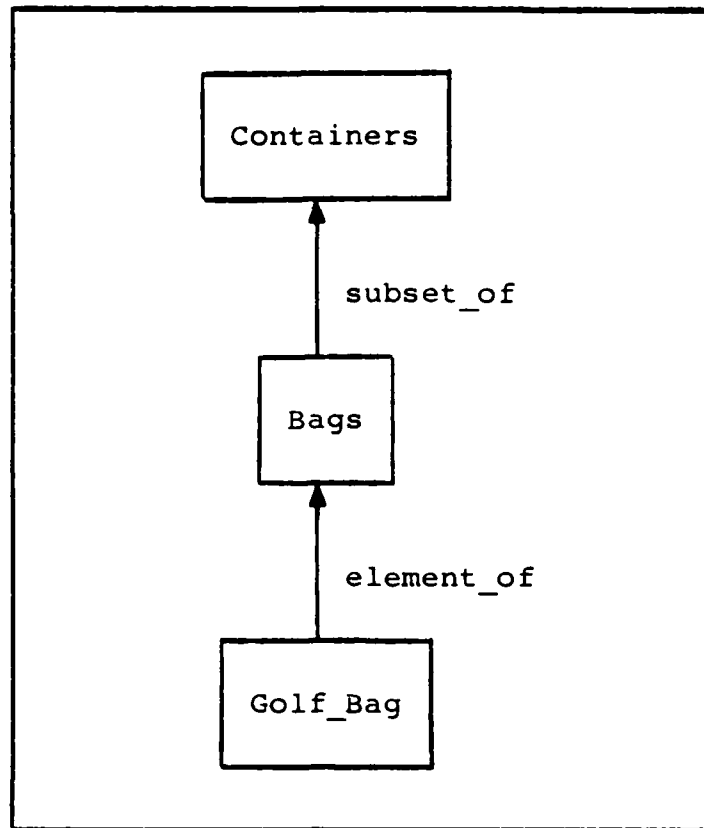


Figure 1. Semantic Network Bag Model

In contrast, neural-network models operate at the micro-level by using various interconnection patterns of neurons to model functions. For example, the word Bags could be modeled in a neural network as shown in Figure 2. Each letter symbol is modeled by a single neuron. To operate, the letter-neurons each fire when stimulated (e.g. reading the letters). By receiving the proper electrical inputs, a word-neuron at a second level is then stimulated and fires. This word-neuron could then be the same word, Bags, used in the semantic network of Figure 1.

From Figure 1 and Figure 2, interesting analogies can be drawn. One is that the micro-level model is analogous to

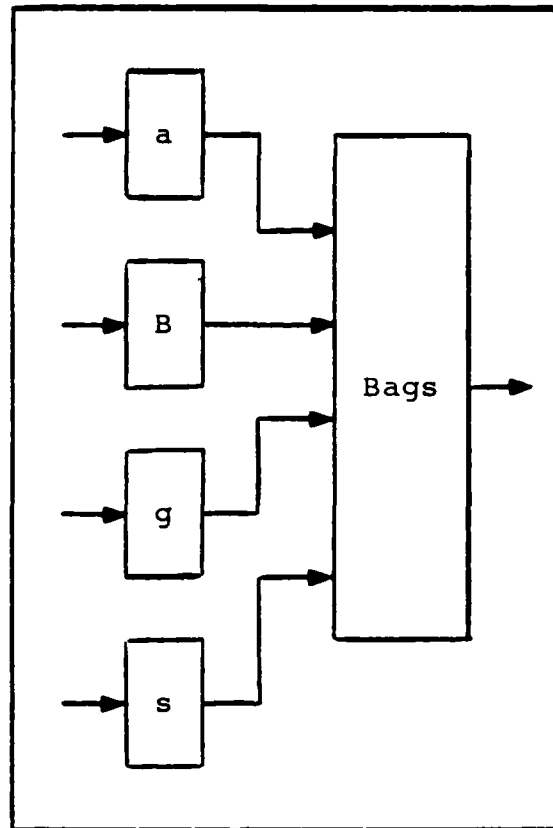


Figure 2. Neural Network Word-Model

the hidden implementation details of an abstract data type (3). Another is that the macro-level models which deal with memory modeling (see Table I) are analogous to atomic abstract data type constructs. Another analogy is that any macro-level model which uses a memory model is analogous to an object level computer program (4). These analogies are not unexpected since cognitive models have been implemented on computers.

Artificial Intelligence Definition. Because of this relationship between AI and psychology, extremists' views have caused difficulties in defining artificial intelligence. Some argue, without proof, that these two fields are

indeed equivalent. That is, humans are equivalent to logic devices. Others argue that the fields are entirely unrelated because human creativity does not always follow a logical path. While these different positions exist, the proponents of each position are really arguing the existence of a single model which would account for human intelligence.

To keep from being involved in the issues surrounding the existence of a single human intelligence model, a definition of AI which handles both extremes is used. While this definition is a combination of ideas from the AI handbook (2; 5), it is not so stated within the body of that text. The definition of AI used here is as follows:

Definition 1.01: AI is the study of encoding human knowledge into abstract data types and the development of computer procedures to manipulate the encoded knowledge into an output form which, when decoded, exhibits aspects of intelligent human behavior.

This definition encompasses the views of both sets of extremists as a result of the encoding of the knowledge and the decoding of the output. If humans are logical devices, then the encoding/decoding is a moot point. If not, then the level of the behavior exhibited rests entirely on the encoding/decoding external to the machine.

In addition, this definition is capable of encompassing the specialty areas of planning, theorem proving, problem

solving, learning, knowledge representation, robotics, pattern recognition and expert systems within the field of AI research. For example, the manipulating procedures identified in Definition 1.01 can be considered as the procedures used in the problem solving models as given in Table I.

Formal Language Theory. Besides these relationships among AI, psychology and computer programs, there is another field where AI associations are strong: formal language theory. To better understand these ties, some background on formal language theory is presented. Because formal language theory is well developed, only a very limited background is given. The reader is referred to text books such as Revesz (6) or Ginsburg (7) for detailed background and to Appendix A for additional formal language references.

Definitions and Characteristics. Formal language theory received attention when Chomsky proposed his four types of grammar (8). In general, a grammar is defined as follows (6:2):

Definition 1.02: A generative grammar is an ordered

four-tuple,  $G:=(NT,T,S,P)$ . NT is a finite set of nonterminal symbols. T is a finite set of terminal symbols (alphabet). S is a special nonterminal start symbol subset contained in NT. P is a set of Post rewrite production-rules of the form  $a \rightarrow b$ .



Each production rule in  $P$  takes a single nonterminal symbol and rewrites it into a new combination of terminal and/or nonterminal symbols. The production rules are applied nondeterministically beginning with the start symbol and ending when the string (word) being generated contains only terminal symbols. A single grammar is capable of generating all the possible words in a language.

Chomsky's four types of grammar are defined via variations of their production rules.

Definition 1.03: A type-0 grammar,  $G(0) := (NT, T, S, P)$ , is a generative grammar which contains unrestricted production rules of the form  $a \rightarrow b$  where  $a \in (NT \cup T)^+$  and  $b \in (NT \cup T)^*$ . The "+" and "\*" indicate the Kleene closure of the elements of the indicated set without the empty word and with the empty word, respectively (8:142).

Definition 1.04: A type-1 grammar (context-sensitive),  $G(1) := (NT, T, S, P)$ , is a type 0 grammar with the added restriction that each string produced after applying a single production rule must be greater than or equal to the length of the string before the application of that rule. That is,  $mjv \rightarrow mkv$  where  $j \in NT$ ,  $k \in (NT \cup T)^+$  and  $m, v \in (NT \cup T)^*$  (8:142).

Definition 1.05: A type-2 grammar (context-free),

$G(2) := (NT, T, S, P)$ , is a type 1 grammar which has only productions of the form  $j \rightarrow k$  where  $j \in NT$  and  $k \in (NT \cup T)^+$  (8:142).

Definition 1.06: A type-3 grammar (regular),

$G(3) := (NT, T, S, P)$ , is a type 2 grammar which has all production rules in one of two forms:  $j \rightarrow pn$  or  $j \rightarrow p$  where  $j, n \in NT$  and  $p \in T$  (8:142).

The inheritance of restrictions shown explicitly in the previous definitions of the grammar-types was proven by Chomsky:

Theorem 1.01: Type 0  $\supset$  Type 1  $\supset$  Type 2  $\supset$  Type 3 (8:152).

Chomsky also proved that the type-0 languages are equivalent to recursively enumerable sets (the set of languages accepted by Turing machines). Chomsky and others continued research and showed that type-1 languages are a subset of the recursive sets. Also, type-2 languages were proven to be those languages accepted by a nondeterministic, finite machine with one pushdown-store. In addition, type-3 languages were found to be regular sets (languages accepted by a finite automaton) (9:43).

Languages and Grammars. From these initial definitions and characteristics, language theory research spread out in several directions. In one direction, many languages were defined and studied. Examples include real-time languages (10), one-way list-storage languages (11), time-

bounded languages (12), tally languages (13), counter languages (14), left-derivation bounded languages (15) and languages constructed from the sentential forms (interim words containing both terminals and nonterminals) called Szilard languages (16).

One of the most studied classes of languages is the regulated languages. These languages use a context-free grammar along with a set of rules that control when a specific rewrite production-rule can be applied. These languages are part of the set used for compiler applications. Examples of these languages include those generated by programmed grammars (17), scattered context grammars (18), attribute grammars (19) and indexed grammars (20).

In another research direction, grammars were studied abstractly. Some examples are grammar schemata (21), compound grammars (22) and grammar forms (23).

Not being satisfied with studying individual varieties of languages, language researchers found certain common characteristics existed among different languages (e.g. closure under intersection with regular sets). These findings led Ginsburg and Greibach to develop the concept of Abstract Families of Languages (AFL) (24). An AFL is a collection of sets of finite words (languages) each of which was generated from some finite alphabet. By definition, an AFL is closed under union, product, empty-word free Kleene closure, inverse homomorphism, empty-word free homomorphism and intersection with regular sets.

One of the unique characteristics of AFLs is that the contained languages can be generated from different alphabets by different types of grammars. Note that regular sets, context-free languages, context-sensitive languages, recursive sets and type- $\emptyset$  languages, all over the same alphabet, each form an AFL by itself (24).

While an AFL is restricted to the six closures specified, the study of AFLs, in general, is not bounded by those closures. For example, a full-AFL is one which is closed under all possible homomorphisms (25). An additional example is the pre-AFL where the member languages do not meet some of the six closure properties (26). On the other hand, a super-AFL is a full-AFL which exhibits additional closure relationships beyond the six specified (25).

While AFLs contain many languages, special families exist where the theorist only has to study a single language in order to obtain the characteristics of the entire family: principal-AFLs (27). In these families, a single language exists from which all other members can be obtained by applying the six closure operations on this "base" language (24). The closure relationships pass the characteristics of the base language to all descendants.

In addition to studying specific families of languages researchers investigated hierarchies of languages. For example, ordering languages by the time needed to accept them forms a language hierarchy (28). A different language hierarchy is formed by ordering the languages based on the

number of nonterminal symbols allowed on the right hand side of the rewrite production-rules (29).

Perhaps one of the most important language characteristics that AFL theory has yielded is the existence of infinite hierarchies of families of languages. For example, an infinite hierarchy of context-sensitive AFLs has been shown to exist (30). Another infinite hierarchy of AFLs can be generated by applying the substitution operator to any full-AFL that is not closed under substitution (25). Even infinite hierarchies of AFLs, each of which is an infinite hierarchy of AFLs, has been shown to exist (31).

Language Problems. Formal language theory involves more than the study of different varieties of languages. It also involves the study of the solvability of language related problems. For example, given two type-3 grammars, it is possible to determine if they generate the same language without having to enumerate all the words in each language and compare them (weak equivalence). However, for type-2 grammars, this is not possible (unsolvable)(9:54-67).

Since some cases of weak equivalence were found to be unsolvable, variations on the equivalence definition were made in the search for solvable equivalence. One example variation for language equivalence is structural equivalence. Structural equivalence exists when languages use the same derivational tree structure (32). An example variation for grammar equivalence is transformational equivalence. There, grammars which are able to be translated into one

another by application of homomorphisms are considered equivalent (33).

Other investigations involved the study of the solvability of the word membership problem. That is, proving that a word belongs to a given language. For example, given a word and a grammar, it is unsolvable for any type-0 grammar whether or not the word is in the language generated by the grammar (9:63). The word membership problem, however, is solvable for type-1 grammars (9:54).

Because the existence of these unsolvable problems caused concern for those investigating using languages to model other problems, language theorists searched for special classes of languages where these problems could be solved. For example, context-free grammars which have rewrite production-rules that can be used either forward or backward without changing the set of sentential forms (interim words) have a solvable equivalence problem (34). A similar result was proven for languages generated by parenthesis grammars (35).

Studying language problems as a theoretical curiosity did not satisfy language theorists. Because of the symbolic nature of languages, some theorists teamed with others from different academic fields to attempt to solve problems in those fields by using formal language theory techniques. For example, by using rewrite production-rules in parallel, biological growth could be modeled and studied (36). In another field, graphs were studied via graph grammars (37).

In another example, solutions to mathematical equations were studied via formal language representations (38). A further example is investigation of computer networks and time sharing systems via formal language theory (39).

Relationship of AI and Formal Languages. Given this background on formal language theory, the relationships of AI and formal languages can now be approached. These relationships currently can only be indicated via example cases because no supporting proofs exist in the literature which show that the relationship of formal languages and AI exists for all the AI systems. However, the mere existence of these example cases show that some formal relationships may exist.

To keep from getting involved in the details of these example applications, only a short description of the informal ties between the AI system and formal languages is given initially. Also, a few examples are provided to motivate the development of formal relationships. The reader is referred to the references in Appendix B for additional examples. For more details on the AI systems mentioned, the reader is referred to the AI handbook (2; 5).

In this set of example applications, there are several cases where the tie between AI production systems and formal languages can be seen. One example is the use by Vere of a type-0 grammar to develop a relational production system (40). Another example is the technique used to infer the

control structure for a programmed grammar production system (41).

In addition to these AI production systems, formal languages have several direct relationships with AI learning systems. For example, grammatical inference is considered to be one of the tasks performed by learning machines (42). Formal language theory was also used to prove the nonexistence of an ideal learning machine (43). A further example is a learning robotic system which uses a hierarchy of grammars between the man-machine interface and the robot (44).

Of all the ties between AI systems and formal languages, the relationship between pattern recognition and language theory is the best known. The most familiar example is the use of Web grammars to generate patterns for picture recognition (45). In another case, Attributed grammars have been used in shape recognition (46). A further example is the Plex languages. They were developed to model physical systems like printed characters and circuit diagrams (47).

Recognizing these relationships between AI and formal languages, many questions arise regarding the models of human intelligence that AI computer programs simulate. Are the proposed human intelligence models a sentence in some higher level programming language or an entire programming language themselves? Are there countably infinite versions of a single model? Does a countably infinite number of unique human intelligence models exist? Can any model of



human intelligence be proven to truly represent intelligent behavior?

While answers to these questions may be debated for many years to come, one component of these models can be investigated in regard to these questions. That component is the form of knowledge representation used to simulate the human storage of knowledge.

Individual Knowledge Representations. In general, knowledge representations can be considered as abstract data types (3). This characteristic can be identified from the definition of AI (see Definition 1.01). The first part of the AI definition involves encoding human knowledge into abstract data types (standard computer science practice for program data). In the past, AI knowledge representations have been defined as a combination of data structures and interpretative procedures that, if used in the right way in a program, will lead to "knowledgeable" behavior (5). Depending on where one assumes the split to occur between the main program's control procedures and the data structure's interpretative procedures, the abstract data types referred to in the AI definition can be considered to coincide with the definition of knowledge representation. The results of this dissertation are based in part on this separation.

Because of this abstract nature of AI knowledge representations, there is a wide variety of abstract data types that compete for membership in the class of AI knowledge

representations. In general, a representation is referred to by the name of its underlying data structure and the data-structure accessing procedures are usually called the inference mechanisms. In this subsection, examples of the more popular individual representations are presented. The discussion on each representation contains, in order, descriptions of the representation's structure, the applicable inference mechanism and a specific example. For the sake of brevity, only a limited description of each representation will be provided. Cited references can provide the reader more detailed descriptions. For more information regarding such areas as applications, advantages/disadvantages, variations, etc., the reader is referred to reference 5 and to the references contained in Appendix B.

Predicate Calculus. Logic has been one of the strongest foundations for philosophers over hundreds of years. This strength comes from the consistency that logic provides as a formal mathematical system. These same characteristics lead AI researchers to consider logic as a knowledge representation scheme.

Of all the different variations of logic (e.g. modal, intuitionist, predicate), AI researchers selected predicate calculus as a candidate representation. Because of the hierarchy of predicate-calculus types (e.g. propositional calculus, quantified propositional calculus, equality calculus, first-order predicate calculus, second-order

predicate calculus), the name "predicate calculus" is insufficient to describe the representation (9: Chapter 2). The actual representation uses first-order predicate calculus with equality.

The primary reason that first-order predicate calculus was chosen is that Church proved that the validity of first-order predicate calculus statements is partially solvable while for second-order predicate calculus statements it is not solvable (48). This means that algorithms exist which will verify that true first-order statements are indeed true without having to generate all possible true statements.

Secondary reasons for the selection of first-order calculus involve the "expressibility" of the representation. The n-ary function constants provide the capability to represent more than just true/false knowledge. For example, the function Father(Joe) can return the name John. Equality provides the capability for function results to be tested as well as the normal true/false values. For example, the test Father(Joe)=Bill? returns false.

As noted previously, a knowledge representation scheme's usefulness comes from the ability to retrieve information from the structure via the inference mechanisms. Robinson's computer algorithm based on the resolution principle provides the primary inference mechanism for first-order predicate calculus (49). His algorithm applies individual procedures that generate new true statements from previous

$$(Vx)[\text{Clubmember}(x) \Rightarrow \text{Fathename}(\text{John}, \text{Father}(x))]$$

Figure 3. Predicate Calculus Example

ones in an ordered manner. This ordered approach provides a capability to obtain a solution to a query without generating all true statements. However, Robinson's approach is still combinatorial in nature so the number of true statements the algorithm generates is not necessarily the absolute minimum.

Figure 3 contains an example of the following statement in first-order predicate calculus: all members of the club have a father named John. The variable  $x$  can represent a club member name or ID number. The predicate constant, `Fathename`, returns true if the function, `Father`, returns the name John. The equality test is embedded within the `Fathename` predicate.

Production Rules. One of the most influential developments in mathematics of this century was the production-rule system introduced by Post (50). Moreover, psychologists had been modeling human actions as a series of stimulus-reaction pairs. Recognizing the relationship of Post's system to these models, Newell and Simon developed production-rule models of human intelligence (51).

In these rule-models, a production rule consists of an antecedent-consequent pair. This pair is tied together in the form of an implication statement. That is, if the

antecedent is true then the consequent is true. The antecedent-consequent pair may contain declarative knowledge. However, the pair is not restricted to that type of knowledge. The consequent part may be complex procedures. With the added concept of "procedural attachment," the antecedent part may also entail execution of a procedure before determining its truth value.

Even though knowledge is represented in a logical formalism for which a limited set of the predicate calculus inference mechanisms would apply, production-rule inference mechanisms are just a restricted form of "generate and test" algorithms. That is, instead of generating a rule, these algorithms search the already established rule-set and test for rules which have their antecedent condition met. The most popular inference mechanisms are backward-chained and forward-chained search (5).

One of the unique features of production-rule systems is the close tie the individual rules have with the working memory during operation. Production rules only represent conditions that could be true about a given situation. For example, the set of production rules may contain one rule that has the consequent that a fact is true while another rule may have the consequent that the same fact is false. To determine the correct answer for a given situation, a production-rule system has to make use of a working memory that contains information unique to the particular situation. That is, the set of production rules alone may not

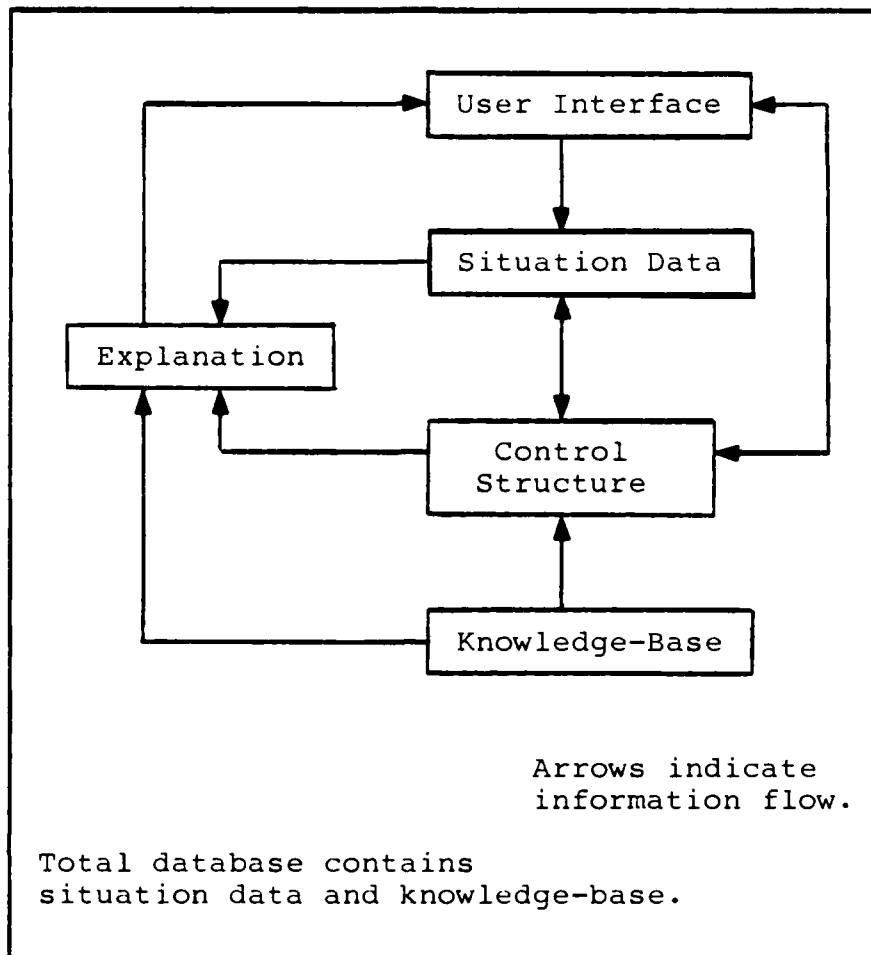


Figure 4. Expert System Functional Diagram represent the complete knowledge about an event. A production-rule system has to use a working memory in conjunction with the production rules to reach a conclusion. This working memory has to contain the information needed to determine which antecedent is true so that the correct consequent can be applied for the given situation.

To better understand the relationship of this working memory in a production-rule system, Figure 4 shows a functional diagram of an expert system. The production rules reside in the knowledge-base and the working memory is

```
If (?patient fever>104) and
    (situation data contains patient lab results)
Then
    (assign lab results to patient) and
    (call rule 37)
```

Figure 5. Production Rule Example

contained in the situation data. Therefore, as the expert system operates, the control structure uses both the production rules and the situation data to eventually reach a conclusion.

Because of this close tie with the working memory, individual production rules are allowed to interact with the working memory. This interaction can involve viewing the contents of working memory as an entity and modifying the contents of the memory as needed.

Figure 5 contains an example production-rule representation for a medical production-rule system. While the rule does not represent a simple English language statement, it does reflect the tie of the working memory to the production rules as well as the complexity of the rules. In the first antecedent clause, the unit variable "?patient" allows for the use of the rule with a specific patient's name. In some implementations the fever value could be an attribute assigned to the specific patient's name. Then, the "greater-than" computation could be accomplished by a procedure call that retrieves the fever attribute-value pair and performs the greater-than test. The second antecedent

clause allows the situation data to be searched for the list of lab results. This list can be entered separately by lab technicians that are not directly involved in the execution of the overall production system.

Once these two antecedent clauses are determined to be true, the consequent is activated. The first clause of the consequent assigns the lab results as attribute-value pairs to the patient's name. This is actually a procedure call. Then, the second clause directs the inference algorithm to select the next rule to be tested.

Semantic Networks. The semantic memory model (see Figure 6) developed by Quillian was an attempt to handle word meaning in a computer by using a hierarchy of syntax pointers similar to a dictionary (52). The model was made up of nodes connected by directed arcs (associative links). The nodes consisted of two kinds: type (TP) and token (TK). A "type" node contained a single word whose definition was obtained by following the associative links attached to it to other word nodes. In memory, only a single type-node was assigned to each word. In Figure 6, the type-node TP1 represents the definition of word A. By following the links, two other nodes are reached that contain words B and C. These two additional words provide the definition of word A.

The "token" node was a place holder for words that were used in the definition of another word. From a given type-node, associative links were connected to token-nodes.



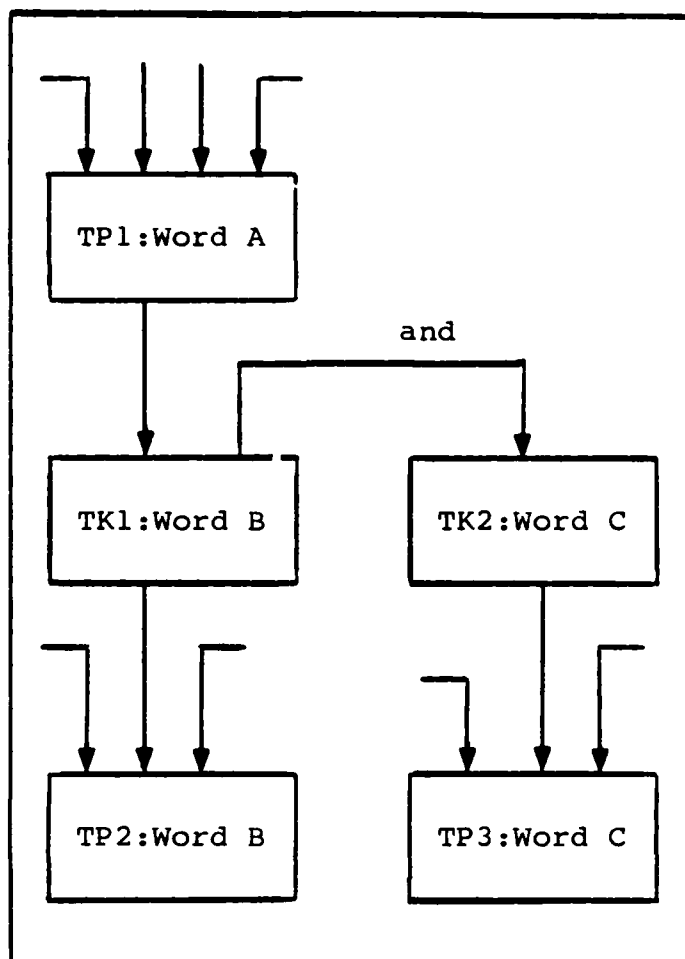


Figure 6. Quillian's Semantic Memory

To keep from reproducing the type-node and all its links every time a word was used in a definition, a token-node was used which pointed to the single type-node for this embedded word. Therefore, there may be many common named token-nodes throughout memory but each points only to its respective, single, type-node. In Figure 6, nodes TK1 and TK2 represent the token-nodes for words B and C. By following the links leaving these token-nodes, their type-nodes, TP2 and TP3, can be reached.

The associative links that interconnected the nodes handled a limited set of primitive relations among the nodes. These included set/subset, a form of attribute-value representation, the token-to-type pointer, conjunction and disjunction. More complex relationships between two nodes were handled by dual associative links with the relationship name residing in the node.

The inference procedures associated with the network stem from Quillian's spreading activation procedure which "drew" inferences about two concepts (words). His procedure is based on the inheritance capability of the network. The procedure "activated" the two type-nodes representing the concepts. Their links were then followed to other nodes. These nodes were activated and their links were followed. The process was repeated until a common node was activated by following paths from both concept-nodes. The common node then described the connection between the original concepts.

From this beginning, the knowledge representation of a semantic network evolved (53). Nodes now represent words, concepts, objects, etc. With the advent of procedural attachment, the nodes can represent calls to selected procedures. The associative links now represent many complex relationships (54).

In the simplest form, the inference procedures for a semantic network operate on a basic matching routine. The routine constructs a data structure that represents the information being sought. Then, the semantic network is

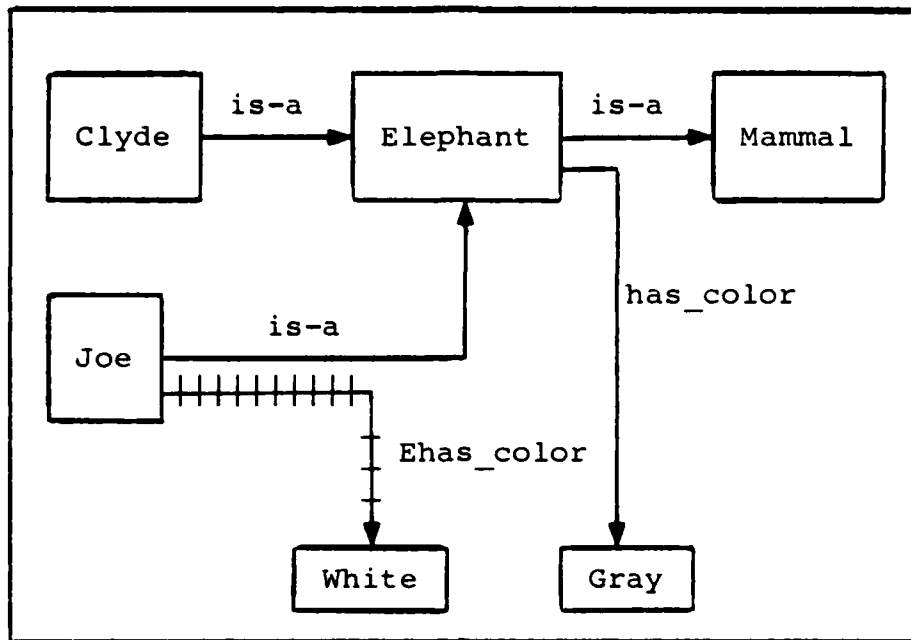


Figure 7. Semantic Network Example

searched for a matching structure and the information is retrieved. Exact matches are not necessary since the inheritance capability can be used to locate common information that may be stored at a higher level in the network.

Unlike the production rules, a semantic network contains information that is always true independent of a given event. The working memory (situation data of Figure 4) provides only a local storage function for the query and its data structure equivalent. In effect, a semantic-network representation contains all the true information about the world that it represents. On the other hand, production rules are usually combined with the situation data to generate world truths.

Figure 7 contains an example of a semantic network representation of the statement: Clyde is a gray elephant,

Joe is a white elephant and both are mammals. The associative links "is\_a" show the relationship of Joe and Clyde to elephants and the inherited relationship of mammals. The "has\_color" link shows the normal color of elephants. The "Ehas\_color" link shows the exception link indicating that Joe is white instead of gray. In the operation of a semantic-network system, the exception link would dominate the inherited has\_color link for a query involving Joe.

Frames. Frames were originally proposed quite informally by Minsky to handle the evidence that people use previous experience to interpret new situations. To use Minsky's own words (55:212):

A frame is a data-structure for representing a stereotyped situation, like being in a certain kind of living room, or going to a child's birthday party.

For each generic type of frame (room, party, automobile), slots exist which are filled either with pointers to other frames, terminal data or default data (see Figure 8). The frame may even contain information about how to use it or what to do if it does not exactly fit the situation. Essentially, frames are tables of relations where the values of the attributes are either explicit or implicit.

The inference mechanisms are basically pattern matching operations. A frame is selected which matches the input requested information. In case no exact match is possible, a metric such as the maximum number of slots matched may be used so that at least a related but not necessarily correct

frame may be found. Using the information contained within the selected frame (e.g. sequence to follow, inheritance pointers, defaults, procedural attachment), the answer to a request can then be generated.

Figure 8 contains an example frame representation of the statement: Joe lives in a two-story frame house at 955 Mann, Larned, Kansas and the house has 1000 square feet and cost \$50,000. To incorporate this information into the frame-system, a house-frame would be selected and filled in as shown. Since there is no missing data in this example, the only default occurring is the ancestor slot in the house-frame. The square foot cost slot is a procedure call which can compute the value if it is needed. See reference 56 for several examples of frame-systems shown pictorially.

Scripts. In the Minsky paper on frames, the work of Schank and Abelson was characterized in relationship to frames (55). In reaction to this characterization, Schank and Abelson countered that the frame description was too general for use. They proposed that their work be considered as a specialization of a frame idea. They called their approach "scripts" (57).

Scripts define a sequence of events which normally occur in association with another event. The idea is that if a given event can be predicted with regularity, only the variations from the norm need be identified in a definition of the event. Therefore, the stored information is a

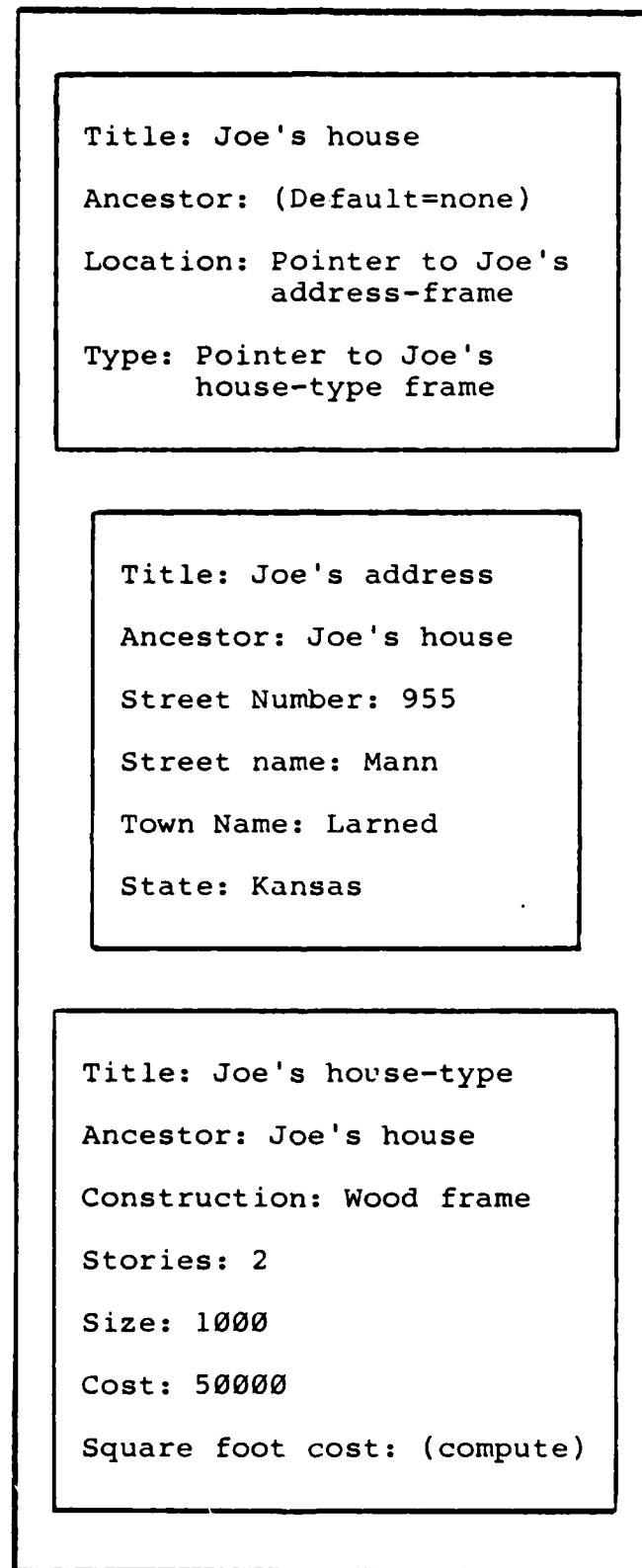


Figure 8. Frame Representation Example

generic sequence of events which contain certain default slots for variations.

The inference mechanisms for scripts are the same as for frames for the initial frame selection. However, once the frame is selected, the system remains within that frame for all information retrievals. A sequence of events is followed to obtain the requested information.

The restaurant script is the most famous example of this knowledge representation scheme. Figure 9 contains a limited version of this script. Even though this is a simple example, statements like "Joe ate a hamburger in the restaurant" will instantiate the script. Then questions like "did Joe pay the check?" can be answered by investigating the departing sequence contained in the script. The defaults in the departing sequence allow for a case when the food is bad and Joe refused to pay the check.

Conceptual Dependency. As with all the knowledge representation approaches, the goal is to represent knowledge in a machine usable form. In addition to the machine-form, the question arises as to the form of the knowledge itself. Since knowledge is usually described by "natural" language, initial research into the primitive structure of natural language was conducted by Schank and Tesler (58). From this early work, Schank developed a conjecture known as "conceptual dependency" (59).

Schank's conjecture was that the "meaning" obtained from a natural language statement can be represented

```

Title: Restaurant Script

Customer: (default=customer)

Players: Waitress, Chef,
         Cashier

Entry: 1. Customer enters
       2. Customer waits to
         be seated
       3. Customer sits down
       4. Waitress gives
         customer a menu

Order: 1. Waitress arrives to
         take order
       2. Customer orders
         (default=hamburger)

Eating: 1. Waitress delivers
         order
       2. Customer eats

Depart: 1. Waitress brings
         check
       2. (default=customer
         leaves tip)
       3. (default=customer
         pays cashier)
       4. Customer leaves

```

Figure 9. Script Example

(defined) by a primitive set of actors (ACTs) acting on the objects in the statement. Hence, equivalent sentences would translate into the same combination of ACTs.

While the number of ACTs is not fixed, the original primitive set consisted of eleven members. Five of the ACTs were associated with physical actions: PROPEL, MOVE, INGEST,



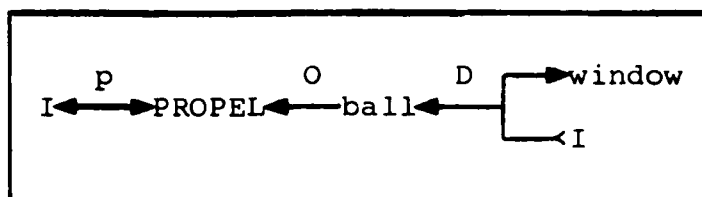


Figure 10. Conceptual Dependency Example (60)

EXPED AND GRASP. Two other ACTs dealt with mental actions: MTRANS and MBUILD. Another two ACTs handled the sight and sound senses: ATTEND and SPEAK. The last two ACTs covered physical and abstract state changes: PTRANS and ATRANS. Later extensions added planning, script and goal actors (60).

The literature does not clearly differentiate between inference mechanisms in general and those used for a specific natural language understanding system. Therefore, the mechanisms are introduced via the operation of a system known as MARGIE (60). In general, natural language statements are translated into the primitive conceptual dependency structure. Then, applying some sixteen different classes of inferences, conclusions are obtained which are either supported by existing facts in the structure or cannot be disproved by facts already in the structure.

Figure 10 contains an example of a conceptual dependency representation of the statement: I threw the ball at the window. The double arrow denotes a mutual dependency between actor and action. The p indicates past tense. The O means the object of the act. The arrow under the O points to the act. The D combined with the "C" shaped arrow

denotes the physical direction of the action; from I to window. The arrow under the D points to the object that is involved in the act.

Combined Representations. AI researchers have found that using only a single knowledge representation, such as one of those described previously, in complex systems is not always advantageous for a given application. For example, a representation may be able to represent all the domain knowledge (Representation Adequacy) but be unable to help direct the attention of the inference mechanism during system operation (Inferential Efficiency) (61:201-202). To try to overcome these types of performance problems, researchers have investigated combining several knowledge representations to gain the advantages each representation can offer. Several examples follow.

A system which combines semantic networks and frames was investigated by Hayes (62). The frame was used as a high-level description of a low-level semantic network. Although more complex in this application, the frame can be seen to represent the "planes" proposed by Quillian (52). Referring to Figure 6, in each type-word plane, the token-words used to define the type-word are obtained by following the links within the plane (filled terminal slots in this combined semantic-network/frame structure). For the token-word nodes, pointers have to be followed to their planes for definition (pointer slots).

In addition to semantic networks, frames have also been combined with predicate calculus. For example, the Krypton representation uses a frame-structure to hold terms that are organized taxonomically (63). Then, first-order sentences are applied to obtain the predicates from the frame-representation. In another example, Charniak proposes a representation that incorporates both first-order sentences and predicates within a frame-structure (64).

Combined representations are not restricted to just the dual systems mentioned previously. For example, a combination of semantic networks, frames and procedures has been used as a method for representing knowledge in an instructional system (65). The frame involves the overall subject while the semantic network shows the relationship of topics. The procedural aspects handle the control of such items as questions, frame control, etc.

In addition to frames and semantic networks, procedural knowledge has also been combined with production rules. For example, a scheme for incorporating procedural knowledge within a production-rule system has been proposed which allows the rules to remain independent of each other (66). A production system can represent the procedural sequences by incorporating the sequences into rules that call rules. That is, the consequent of each rule performs a step and then calls the next rule in the sequence. However, this scheme proposes to use a Recursive Transition Network (RTN)

for the sequencing portions so that the rules remain independent of each other (56:169-170).

Knowledge Representation Relationships. In addition to combining representations, researchers have investigated mapping one scheme into another in search of equivalent representations. There are several reasons that researchers search for equivalence. One reason is they want to show consistency of various representations by mapping them into predicate calculus. Another reason is they want to show that a given representation can "express" the same knowledge as other representations. A further reason is they want to convert one representation into another form so that the advantages of various representations can be used when the particular problem-solving situation warrants. This search for equivalence resulted in the controversial procedural versus declarative knowledge issue (67). While that issue was never resolved, the work involved in attempting to show equivalence of various schemes provides insight into the knowledge representation approaches. Some examples of these mapping efforts follow:

In the previous subsection on individual knowledge representations, the definition of knowledge representation was shown to be analogous to an Abstract Data Type (ADT). While several approaches have attempted to provide ADT definitions for knowledge representations, the most complete one was the semantic network proposed by Dilger and Womann

(68). Their definition required 35 generic axioms to describe an entire abstract network.

Abstract data types were not the only schemes investigated. One mapping that generated the most interest was translating a representation into the predicate calculus scheme. For example, using the KRL-Ø frame-language, Hayes provided a translation of KRL-Ø frames into predicate calculus (69). In another example, Simmons and Bruce developed an algorithm which converts a version of a semantic network into predicate calculus (70).

Predicate calculus formulations were also translated into other representations. For example, the combined work of Sandewall and Hendrix provided an approach to translate predicate calculus into a special form of semantic networks called a partitioned semantic network (71; 72). Sandewall provided the individual clause relationships while Hendrix developed the quantification relationship into a partitioned semantic network.

Hendrix's partitioned semantic network was also involved with translations of production rules. For example, Duda and others translated production rules into a partitioned semantic network (73).

Representations were also translated into production rules. For example, Rychener implemented unique frames via production rules in his Instructable PMS Language system (74). As a concluding example, frames can be used to

implement scripts. This relationship was pointed out by Minsky in his original definition of frames (55).

### The Problem

From this extensive background research on formal language theory and knowledge representations, one major conclusion can be reached. The conclusion is that given all the research heretofore performed by all the philosophers, psychologists, linguists, computer scientists, etc., there still does not exist an acceptable model for human intelligence. Because a human model hinges on our ability to nondestructively investigate the internal workings of a living human being, there are risks that a true model of human intelligence may never be found.

The same risk is passed down to the cognitive scientists in two forms. First, there is a risk that the human cannot be simulated by a computer program. Second, assuming that a computer model exists, there is a risk that it may never be found (see Section IX for more details on these risks).

Because of this lack of a model of human intelligence, there still is no acceptable theory of knowledge representation (5). From the higher level theories of those like Bobrow (75) and Newell (76), through the individual schemes already described previously, no single approach has been found which will handle intelligent tasks. However, knowledge representation research still continues.

Problem Background. As with any computer simulation task where the underlying model is unknown, research on knowledge representations involves the development of experimental versions. These versions will be implemented, evaluated, revised, re-implemented, re-evaluated, etc. Finally, when a representation is actually implemented in an operational system, the selected version will need to be formalized and validated as a general representation scheme.

For a given representation, this validation effort includes providing detailed definition of the representation, determining the scheme's limitations, identifying its computational efficiencies, investigating its relationships with other representations, etc. However, this does not include verification of the model's applicability to humans. The elusive nature of the human model prevents the generation of the necessary verification procedures.

With the AI field's entry into actual real-time applications (e.g. Defense Department's Pilot's Associate Program), knowledge representations are requiring a rigorous evaluation stage. Without the backing of detailed analyses, there are high risks in selecting an acceptable representation for a real-time application. As an example of these risks, a representation could be selected that can only handle a special set of knowledge. But, as the system using this representation ages, new knowledge has to be added to the representation in order to keep the overall system current. When the new knowledge is attempted to be added,

it may be the case that the representation cannot accommodate all the new information efficiently. In the extreme case, the modified system may no longer be able to operate within the necessary time constraints. This could force a complete redesign of the system including selection of a different knowledge representation (61:201-202).

Some of these real-time applications involve expert systems. Associated risks involve the obtaining of the expert's knowledge in a form that can be properly implemented in several knowledge representations. Without supporting comparison analyses, there is a risk that the knowledge gained from an expert may be stored in a form that cannot be effectively translated into other knowledge representations. If this is the case, there is a risk that the particular storage form may only be useful in a limited number (if any) of real-time applications.

While these examples of risks point out the negative aspects of not performing the needed analyses, there are positive side effects that can occur from such analyses. For example, the comparison analysis may reveal the existence of a translation algorithm which could be used to automate the transformation between knowledge representations.

Problem Statement. Even though knowledge representation researchers recognizes this need for rigorous evaluation, they still have not taken the necessary and sufficient steps towards defining, analyzing and comparing knowledge repre-



sentations. Probably the most critical support for the problem statement is that there are no acceptable formal definitions for most of the common knowledge representation schemes. Except for predicate calculus, the individual knowledge representations previously described are not rigorously defined. Because of this lack of definitions, the authors of many published articles begin by providing their own definition of the original representation that they used as the basis for developing their specific representation variant (77; 78). This lack of formal definition has led to the generation of so many variations of these common schemes that it has become difficult to determine where the variation starts and the original definition ends. For example, Brachman addresses the definition issue for semantic networks in reference 79. In the case of production rules, Davis and King point out that the variations have become more an issue of style than differences (80).

An additional support for the problem statement is that while the literature abounds with articles on knowledge representation applications, the majority of these articles fail to present any supporting analysis (see Appendix B for references). Even where some analysis has been accomplished, the results are often in a form that is not compatible for comparisons with the results of other analyses (81; 82). Furthermore, articles identifying and analyzing representation models which failed entirely are notably absent.

Further support for the problem statement is that there is no acceptable common approach for comparing representations. In addition to mapping between representations, there are other approaches for comparing representations such as constructing "expressibility" metrics (83). However, knowledge representation researchers still cannot agree on a basic set of characteristics for comparing representations (84). No existing approach can handle the over one-hundred dimensions suggested in reference 84.

Having identified the problem, the next step is to determine an approach to solve the problem which is done in the next section. Also, the next section contains an overview for the remainder of this document.

## II. Dissertation Approach

### General Concept

Recognizing the lack of analyses and the controversial issues surrounding comparison characteristics, the goal of this dissertation is not to develop another knowledge representation as a solution to the problem. The goal is to develop general insights about existing representations and provide an approach by which future representations can be compared. The conjecture is that formal language theory can be used as the basis from which to generate compatible comparisons of knowledge representations. For the validation tasks associated with knowledge representation (see Section I), there are analogous studies associated with formal language theory. For example, the task of determining the relationships among representations is analogous to the task of determining language/grammar equivalence. Other analogous tasks are the identification of hierarchies of representations (languages) and determining membership of a representation (language) in the hierarchy.

Supporting Rationale. Even though the previous examples in Section I implied the existence of a representation-language tie, stronger rationale supporting the conjecture is required. Therefore, before proceeding with the development of the formal language approach, two additional cases of this tie are examined in detail.

The first case involves Feder's use of a two-dimensional language to describe line pattern images (85). Using this language, he studied the characteristics of the patterns via formal language theory. He used the hierarchical relationships of complement, union and intersection to expand simple, easy-to-study languages into more complex ones. While not presented in detail, he suggests the use of known relationships in formal language theory to determine time and size limitations, solvability of the recognition problem, etc.

The second case involves the formal language relationship of a subset of predicate calculus, known as quantified propositional calculus. Hamblin proved that all the individual statements of quantified propositional logic form a context-sensitive language (86). Therefore, many characteristics of context-sensitive languages could be used in analyzing this subset of predicate calculus. For example, the equivalence problem for a context-free language is unsolvable (9:63). From the Chomsky language hierarchy (see Theorem 1.01 of Section I), context-free languages are contained within the context-sensitive languages. By inclusion, it follows that the equivalence problem for context-sensitive languages is also unsolvable. From this unsolvability characteristic, it follows that it may not be possible to prove that Hamblin's representation is equivalent to any other type-1 representation-language.

The word "may" has to be used in this equivalence conclusion because the equivalence finding for context-sensitive languages is for the general case of all context-sensitive languages. Because formal language theorists have proven the existence of special classes of languages where equivalence is solvable (34), it may be the case that equivalence is solvable for some knowledge representation languages contained in the type-1 language-class. However, Hamblin did not perform any language-class investigations.

Counterpoints. Even with this strong evidence supporting the conjecture, there are several counterpoints to applying formal language theory to knowledge representations. One counterpoint stems from the position that knowledge representations are said to deal with both syntax and semantics (84). The association of semantics with representations cause an "apples and oranges" situation to exist between many representations. An example of this association is represented by the relation between semantic networks and production rules. The semantic-network representation uses symbols to represent all known truths about a given world. But, the production-rule representation uses symbols to represent conflicting truths that can only be resolved by using the working memory (situation data of Figure 4). The fact that the meaning of the symbols represent truths or conflicting truths can only be determined by the semantics assigned to the symbols.

Smith provides an answer to this semantics concern (87). In an attempt to develop a model of human memory based on five levels, Smith defined the concept that everything below the top level is just syntax. The only way to deal with the top level as syntax is to go outside the human environment.

The conjecture of this dissertation is based on the similar position that knowledge representation schemes are syntactical structures within the computer system. The semantics come into play entirely outside the computer. For example, a semantic-network's relation is represented in the computer as memory locations containing symbols (binary equivalent) and memory pointers. The relation name itself has no internal significance except that an internal pointer can be found stored with the name symbols. The "meaning" of a link name is assigned outside the computer system. The computer process that is applied to determine when to follow a path is nothing more than a syntactical pattern match on a symbolic representation of the link's name.

Similarly, a production-rule's antecedent is stored as a set of symbols and symbolically matched against the situation data. The fact that the antecedent may represent, for example, a person's condition (see Figure 5) can only be determined outside the computer system.

Furthermore, conflicting consequents like "patient has disease A" and "patient does not have disease A" are two different atomic symbol strings within the machine. The

conflicting meaning of the strings can only be identified outside of the computer system.

Also, any processes used to interact with the working memory still involve only the primitive machine operations of symbolic comparison and memory handling. By definition, the meaning of the individual contents of working memory can thus only be determined outside the computing machine.

Another counterpoint involves the handling of explicit procedural knowledge within formal languages. Procedural sequences cannot normally be handled by the unordered word generation that a nondeterministic grammar provides. However, depending on how the procedures are defined, there are ways to handle this problem. Procedures can be generalized as any arbitrary computer program over a set of computer programming languages. Thus, the computer programming languages are added into the overall knowledge representation languages. That is, computer programming languages are identifiable subsets of words contained in the overall knowledge representation language.

If the allowable procedures are restricted to a special set of computer programs, then the word order of the implementing computer programming languages becomes important. This word order can be handled in the following manner. Each procedure in the set can be represented as an ordered, finite group of words (with repeats) from the associated implementing computer programming language. For each computer programming language, these ordered groups can then

be considered as a single word in another language. These new languages can then be added to the overall knowledge representation language. Formal language theorists have dealt with procedures defined in this manner via a class of languages called trace languages (88).

A further counterpoint stems from the conjecture that says that most of the knowledge representations are already formally equivalent. This argument comes from using representations to model Turing machines. Since representations can be used to model both the knowledge-base and control structure of Turing machines (see Figure 4), equivalence of representations has already been shown to exist. Hence, there is no need to study the equivalence issue any further. The problem with this conjecture is that the equivalence is determined by comparing the results generated by the overall Turing machine model, not comparing the representations themselves. That is, two different representation forms may be used to model the same Turing machine. However, the two representations used to model the knowledge-base part may not be equivalent syntactically to each other.

This dissertation is based on the position that equivalence of knowledge representations should be determined separately from the rest of the components of a system that uses them. That is, the knowledge-base (see Figure 4) should be viewed from a meta-level where the individual characteristics of the representation can be identified. At this meta-level, the syntax of the representation can be



best studied via formal language theory. Thus, characteristics unique to the representations can be identified regardless of the system implementation. For example, it may be the case that one representation form is easier for users to maintain. However, another form may be more computationally efficient for a certain problem-solving technique. By determining the syntactical equivalences of the representations, it may be possible for users to maintain their representations in one form and perform transformations into other representations when required (centralized database concept).

#### Approach

Having answered the major concerns with the dissertation's concept, the details of the approach can now be discussed. The approach is to analyze knowledge representations by first converting them into a formal language and then proving characteristics of the representations via techniques of formal language theory.

Scope. Because there are many different knowledge representations and variations given in the literature (see Section I and the references in Appendix B), attempting to analyze every one within the scope of this study is an insurmountable task. In order to reduce this effort to a more manageable size, the scope was initially limited to the set of the six individual representations previously

described (predicate calculus, production rules, semantic networks, frames, scripts and conceptual dependency).

Without a formal theory of knowledge representation, the six individual representations cannot be guaranteed to be members of a basis set of knowledge representation. Therefore, analyzing all of them does not necessarily provide coverage of the entire knowledge representation area. But, credibility of the overall approach can be established by using a few of the schemes from this constrained set.

From this set of six schemes, two were then selected: production rules (excluding working memory or situation data) and semantic networks. Production rules were chosen because they are the most widely used knowledge representations in expert systems. Because of this use, the results should be able to assist a large group of expert system developers.

The production system's working memory is excluded as part of the representation for several reasons. One reason is that the contents of the working memory are unique to the particular problem being solved. As previously noted, the representations are being evaluated at a meta-level where system control operation has been separated from the representation.

Another reason the working memory is excluded is that the production rules themselves already contain antecedent/consequent clauses that involve the working memory. The embedded tests for antecedent truthfulness involve symboli-

cally matching the contents of working memory with the contents of a production rule. Hence, the working memory will have to contain duplicate information. Furthermore, the procedures that are called from rules to interact with the working memory are already contained symbolically in a production rule.

Semantic networks were chosen because in addition to being used in expert systems, they are representative of a completely declarative knowledge structure. Also the semantic network is the extreme simplification of a frame. In reference 62, this relationship was explicitly noted where each node (frame) represented a subnetwork. In the limit, each frame could represent a single node with the node name contained in one slot, the relation name in another slot and the link pointer in a third slot. Since part of formal language theory deals with hierarchical characteristics of languages, the semantic network provides the foundation language upon which to expand into the frame hierarchy.

The other knowledge representations were not selected for a variety of reasons. Predicate calculus was not chosen for investigation because it has been studied extensively over the years by others and, as pointed out in the subsection on supporting rationale, the initial effort (formal language mapping and typing) has already been completed by Hamblin (86). Frames were not selected because, as noted previously, frames can be considered to be an extension of

the already chosen semantic networks. Scripts were not investigated because they were shown to be just a specialization of the frame representation (55). Conceptual dependency was not selected because it can be considered as a special layered semantic network.

Overview. With the completion of the justification for limiting the scope to production rules and semantic networks, the details of the actual approach used can now be described. As stated previously, the basic approach is to derive the formal language equivalent for each of the selected representations and then evaluate and compare them via formal language techniques. In order to derive their language equivalent, there are some general results that need to be derived from formal language theory first.

Formal language theory relies on the concept of recursively enumerable sets. Because of the word "enumerable", most theorists take for granted that the size of the language space is countably infinite. There do not exist in the literature direct formal proofs of this fact. Therefore, the dissertation begins, in Section III, with the construction of formal language space and the development of theorems proving the countably infinite size of the space.

With this part of formal language theory established, Section IV focuses on the definitions of the two selected knowledge representations. Because of the numerous variations of production-rule and semantic-network definitions given in the literature, an inclusion hierarchy of charac-

teristics is generated for each representation. The characteristics included in each hierarchy are those used by the knowledge representation community to define particular versions of each representation. Using the lowest level characteristics from each hierarchy, formal definitions for the "category-1" production rules and semantic networks are developed.

By applying these category-1 definitions, a formal language equivalent for each of the two representations is developed in Sections V and VI. Individual characteristics about each of these language equivalents are proven. These include the countably infinite cardinality of each of the languages and their membership in Chomsky's class of type-2 languages. In addition, a word/symbol variation is applied to redefine the specific form of the knowledge contained in these representation-structures. The effects of this variation upon the previously identified type-2 language classification are determined.

In addition to the individual language's characteristics, Sections V and VI also contain proofs of the characteristics of special classes of category-1 languages. These characteristics include the countably infinite cardinality of the classes, the type-2 language-bounds for languages contained in each of these classes, the union and product nonclosure properties of the languages in each of these classes, and the inability of each of these classes to be an Abstract Family of Languages.

With the development of the characteristics of the category-1 languages completed, these characteristics are then used in Section VII to formally compare the two special classes of representation-languages. The comparison techniques used are those for determining equivalence of the representations and determining transformation between the representations (see Section I). For the equivalence case, equivalences between certain subsets of production-rule and semantic-network languages are proven to exist. However, for the arbitrary case, equivalence of production-rule and semantic-network representations at the category-1 level is proven not to exist. For the transformation case, any arbitrary production-rule language contained in the special class of category-1 languages is proven to be transformable into a semantic-network language. However, an arbitrary semantic-network language contained in the special class of category-1 languages is proven to be unable to be transformed into a production-rule language.

Given all of these findings for the production-rule and semantic-network languages, the next step is to expand the representation-definitions according to the inclusion hierarchy developed in Section IV and determine the characteristics for the top-category languages. In Section VIII, the hierarchical expansion is performed with the help of several conjectures. These conjecture focus on the existence of a language equivalent at the top category and the equivalence of the meanings (semantics) of the characteristics tabular-

ized in Section IV. Using these conjectures, an arbitrary production-rule or semantic-network language contained in the top-category classes of languages is shown not to be contained in Chomsky's class of type-2 languages. Additionally, the top-category classes of production-rule and semantic-network languages are shown not to be equivalent. Also, each of these top-category classes is shown not to be an Abstract Family of Languages. Furthermore, an arbitrary top-category production-rule language is shown to be able to be transformed into a semantic-network language. However, an arbitrary top-category semantic-network language is shown to be unable to be transformed into a production-rule language.

Using these characteristics, conclusions are reached in Section IX regarding the use of the knowledge representations of production rules and semantic networks.

Recommendations are then provided in Section X on how to expand and apply the results of this analysis. Suggestions are also presented on how to add additional analysis techniques to the approach. Finally, comments on the contributions of this effort are discussed.

#### Reader's Guidance

Because of all of the interrelated findings of this dissertation, some assistance needs to be provided to help a reader follow through the dissertation and understand the information contained within various subsections. For this

dissertation, this assistance takes the form of a flow diagram, cross-reference lists and examples. Because of the dependence of the theorems and proofs used in one section of this dissertation on those used in a previous section, the dissertation should be read in the order of the section numbers. However, a reader may follow the production-rule or semantic-network language-development alone by referring to Figure 11. The vertical arrows indicate the sequence that should be followed in order to understand a particular section. The horizontal arrows indicate those sections that provide support to a particular section. However, the reader is not required to understand the entire contents of the supporting section in order to understand the particular section that the horizontal arrow indicates.

For example, to understand the semantic-network language-development contained in Section VI, a reader should read Section IV in detail but can refer to portions of the production-rule language-development contained in Section V. However, for the production-rule language-development contained in Section V, Section VI can be ignored.

In addition to this flow diagram, cross-reference lists of definitions, theorems, corollaries, conjectures and propositions are provided in the beginning to help the reader. Many of the proofs refer to other definitions, theorems, etc., by only the item identification number and section. These lists provide the reader a quick page



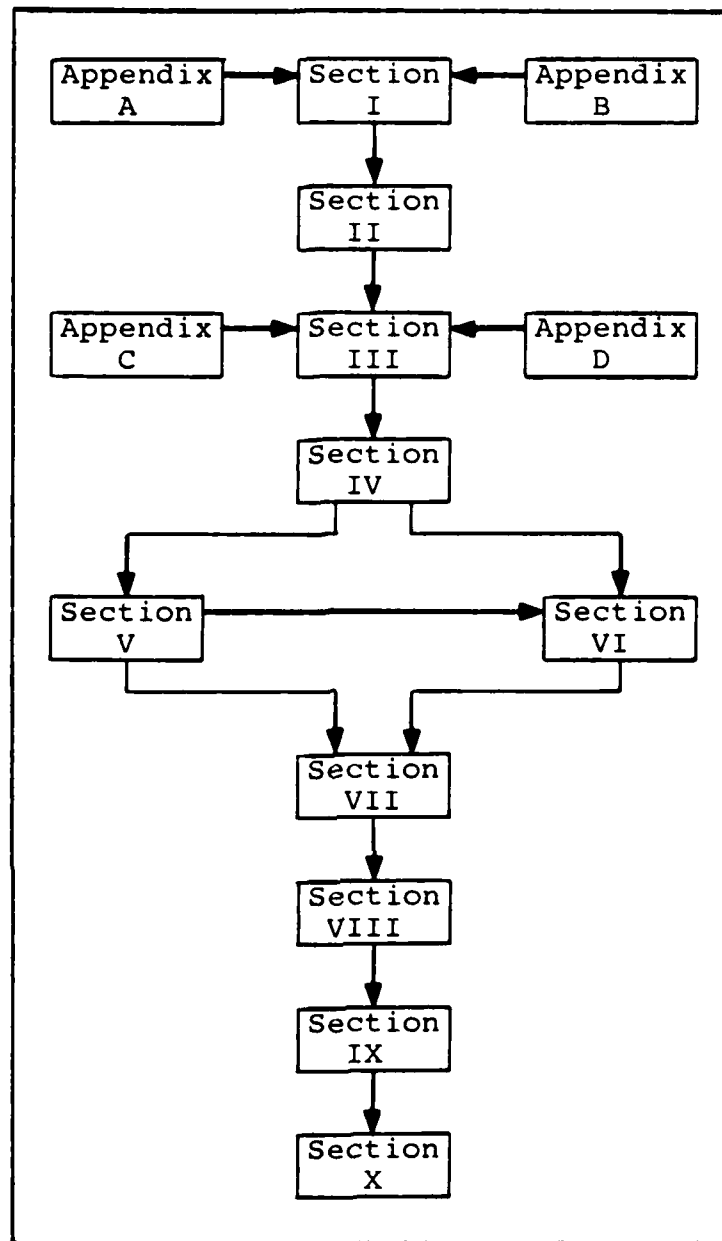


Figure 11. Dissertation Flow Diagram

reference should a review of the identified item be required.

While the flow diagram and cross-reference lists help the reader locate needed information, many example cases help the reader better understand the information that is

presented. For some of the more complex proofs, example cases are presented within the steps of the proof. Even though these examples occur in the proof, the reader is cautioned not to falsely conclude that the proof was done "by example." If a counter-example type of proof is used, this fact is noted at the beginning of the proof.

With this understanding of the document's structure, the reader may now proceed to the first set of findings: the size of language space.

### III. Language Space

After a thorough review of the formal language literature (see Appendix A), many theorists were found to be using several unproven assumptions about the size of languages within the proofs of their own theorems. One assumption they use is that because all languages are recursively enumerable, any infinite length language must contain a countably infinite number of words. Another assumption they use is that any infinite hierarchy of languages must contain a countably infinite number of languages. While these assumptions are not false, the theorists missed related size-characteristics that could only be revealed by supporting proofs of these assumptions. As a result, some of their own theorems have proofs which are more complex than is necessary. Since several findings of this work depend directly on the length and number of languages, these assumptions were rigorously proven so that all the size characteristics could be identified.

Because these assumption proofs depend a great deal on definitions and theorems already provided by other sources, a subsection is given herein which lists these required items. For the theorems of others, their proofs are not provided herein. The reader is referred to the cited reference for those proofs. Even though supporting details are provided in the development, the reader is assumed

to have background in set theory, formal language theory, recursive function theory and Turing machines. Additional information on these subjects can be found in references 8, 9, 89, 90, 91, 92, 93, 94, 95 and 96.

### Supporting Definitions and Theorems

The definitions and theorems contained in this subsection are presented in an order which supports their later use. In order to provide sufficient information to understand some of these definitions and theorems, related definitions and theorems are also given. However, only the main definitions and theorems are referred to in the actual proofs. Therefore, a well informed reader may skip to the next subsection and only refer back to the specific definitions and theorems that are cited in the proofs.

Since the definitions of various grammars were previously given in Section I (see Definitions 1.02-1.06), they are not repeated in this subsection. However, for consistency, the same symbols used in the tuple form of the grammar definition,  $(T, NT, S, P)$ , are also used herein to represent terminals, nonterminals, start symbol and rewrite production-rules, respectively. Because the terms "word" and "language" are associated with grammars, their definitions need to be provided at the outset.

Definition 3.01: Let "Z" be any finite set of symbols called an alphabet. These symbols are called the letters of the alphabet (9:2).

Definition 3.02: A word (string) over  $Z$  is any finite, concatenated sequence of letters from  $Z$  (with repeats allowed) (9:2).

Definition 3.03: The empty word, " $\epsilon$ ", is the word consisting of no letters (9:2).

Definition 3.04: The closure of  $Z$ , " $Z^*$ ", denotes the set of all words over  $Z$  including  $\epsilon$  (9:2).

Definition 3.05: The empty set, " $\emptyset$ ", is the set containing no words (9:2).

Definition 3.06: A word, " $w$ ", that is an element of  $Z^*$  is generated by a grammar, " $G$ ", if there is a finite sequence of words over  $N \cup Z$  such that  $w(0)$  is equal to a symbol  $s \in S$  and  $w(n)=w$  and  $w(i+1)$  is obtained from  $w(i)$ ,  $0 \leq i < n$ , by application of one of the production rules in the production rule set  $P$ . The terminal set of  $G$  is equal to  $Z$  in this definition (9:41).

Definition 3.07: The set of all words that can be generated by  $G$  is called the language generated by  $G$  (9:41).

Now that a language has been defined as a set of words, set-theory definitions and theorems are necessary for further refinement.

Definition 3.08: Let  $A$  be a set. The power set, " $PS$ " of  $A$  is the set of all subsets of  $A$ , including  $A$  and  $\emptyset$  (90:92).

Definition 3.09: Cardinal numbers are the numbers used to measure the size (number of elements contained within) of a set (90:279).

Definition 3.10: A set is said to be finite if its cardinal number is a natural number. A set that is not finite is said to be infinite (91:103).

Theorem 3.01: The set, "N", of natural numbers is an infinite set (90:276).

Definition 3.11: Aleph null, " $N_0$ ", represents the cardinality of N (90:280).

Definition 3.12: A set A is of cardinality  $N_0$  if there is a bijection (see Definition 3.15) from N to A (90:280).

Definition 3.13: A set A is countably infinite if the cardinality of A is  $N_0$ . A set is countable if it is finite or countably infinite. The set A is uncountable or uncountably infinite if it is not countable (90:280).

Theorem 3.02: The subset of real numbers over  $[0,1]$  is not countably infinite (90:284).

Definition 3.14: A set A is of cardinality "c" (uncountably infinite) if there is a bijection from  $[0,1]$  to A (90:286).

Theorem 3.03: Let A be a finite set. Furthermore, let " $|A|$ " denote the cardinal number of A. Then,  $|A| < N_0 < c$  (90:291).

Theorem 3.04: Any subset of a countable set is countable (91:39).

Theorem 3.05: The union of a countable set of countable sets is countable (91:41).

Theorem 3.06: The set  $Z^*$  is countably infinite for any finite alphabet  $Z$  (90:282).

Theorem 3.07: If  $Z$  is a finite, nonempty alphabet, then the cardinality of  $PS(Z^*)$  is  $c$  (90:285).

Theorem 3.08: Every infinite set contains a countably infinite subset (90:282).

Definition 3.15: A one-one correspondence between two sets exists if a pairing of the elements of either set with the elements of the other set exists. A bijection is a one-one correspondence (91:36).

Definition 3.16: Two sets,  $A$  and  $B$ , are said to be equipotent if and only if there exists a one-one correspondence between them (91:37).

Theorem 3.09: A necessary and sufficient condition for a set  $A$  to be infinite is that there exist at least one proper subset of  $A$  which is equipotent to  $A$  (91:104).

Theorem 3.10: The set of all finite subsets of a given countably infinite set is countably infinite (91:42).

This completes the basic supporting information necessary to establish characteristics of a given language. But,

additional theorems and definitions are required to establish the relationship of languages. One of the required theorems is the Chomsky inclusion hierarchy of languages. Since the Chomsky inclusion hierarchy was given Theorem 1.01 of Section I, this theorem is not repeated again. However, specific definitions and theorems regarding languages and Turing machines are provided to complete the background for the language size proofs.

Definition 3.17: A set of words,  $A$ , over a finite alphabet,  $Z$ , is recursively enumerable if there is a Turing machine which accepts every word in  $A$  and either rejects or loops for every word in  $Z^* - A$  (9:38).

Definition 3.18: A set of words,  $A$ , over  $Z$  is recursive if there is a Turing machine which accepts every word in  $A$  and rejects every word in  $Z^* - A$  (9:39).

Definition 3.19: A set of words over  $Z$  is a regular set if the set is generated recursively by a finite number of applications of the following steps:

1. Every finite set of words over  $Z$  (including  $\emptyset$ ) is a regular set.
2. If  $U$  and  $V$  are regular sets over  $Z$  so are their union and product.
3. If  $U$  is a regular set, so is its closure,  $U^*$  (9:2).



Definition 3.20: A set of words over  $Z$  is called a type- $i$  language ( $i=0,1,2,3$ ) if it can be generated by a type- $i$  grammar (8:143).

Theorem 3.11: A set of words over  $Z$  is recursively enumerable if and only if it is a type-0 language (8:143).

Definition 3.21: A set is regular over  $Z$  if and only if it is accepted by some finite automaton over  $Z$  (9:7).

Definition 3.22: A function  $f:Z^* \rightarrow Z^*$  is Turing realizable if there exists a Turing machine,  $M$ , such that  $M$  has a halted computation  $(e, qI, \#, w) \Rightarrow (\phi, q, \#, v)$  if and only if  $f(w)=v$ , and  $M$  fails to halt for an input tape,  $\#w$ , if and only if  $f(w)$  is undefined (92:445).

Definition 3.23: Let  $N$  represent the natural numbers. A function  $f:N^m \rightarrow N$  is Turing computable if there exist a Turing machine that realizes the function  $g:\{\#,1\}^* \rightarrow 1^*$ , where  $g(1^{x_1}\# \dots \# 1^{x_m}) =$

$$\begin{cases} 1^y & \text{if } f(x_1, \dots, x_m) = y \\ \text{undefined} & \text{if } f(x_1, \dots, x_m) \text{ is} \\ \text{undefined} & (92:463). \end{cases}$$

Definition 3.24: A subset,  $K$ , of the natural numbers,  $N$ , is Turing enumerable if either:

$$1. \quad K = \phi$$

or

2. There exists a Turing computable function,  $f:N \rightarrow K$ , that maps  $N$  onto  $K$ . We say  $f$  is an enumeration of  $K$  in this case. Using Godel numbering this can be extended to an arbitrary set  $X \subseteq Z^*$ , where  $Z$  is any alphabet. Set  $X$  is Turing enumerable if the set  $\{Gd(w) | w \in X\}$  is Turing enumerable.  $Gd$  is the Godel numbering of  $Z^*$  (92:487).

Theorem 3.12: Type 3 languages are the finite state languages (8:150).

Theorem 3.13: There exists a Turing computable function,  $W: N \times N \rightarrow N$ , such that for all  $x, y, z$ ,  $W(z, x) = y$  if and only if machine  $M(z)$  in an enumeration of Turing machines computes  $f(x) = y$  (92:491).

### Language-Space Size

With this background established, the size of language space can now be determined. First, the space needs to be constructed using some of the definitions and theorems of the previous subsection.

Let  $Z$  represent an arbitrary finite alphabet per Definition 3.01. Then, applying Definitions 3.02, 3.03, 3.04, 3.05, 3.06 and 3.07 and Theorem 3.11, let "L" be defined as a set which contains all the languages generated by all

possible grammars over  $Z$ . That is,  $L$  contains all type-0 languages over  $Z$ .

Next, applying Definitions 3.04 and 3.08, let " $D$ " be defined as a set that contains all possible subsets of  $Z^*$ . That is,  $D = \{PS(Z^*)\}$ .

By these definitions,  $L \subseteq D$  since  $L$  is also a set containing subsets of words from  $PS(Z^*)$ . At this point, it cannot be determined whether or not  $L$  is a proper subset of  $D$ . However, a size characteristic can be established for each formal language in  $L$ .

Theorem 3.14: The cardinality of every set of words  $d \in D$  is countable.

Proof: From Theorem 3.06, the set of all finite words  $Z^*$  is countably infinite. Since  $D$  contains all possible subsets of these words, Theorem 3.04 can be applied to conclude that each element in  $D$  is countable. Q.E.D.

Corollary 3.01: The cardinality of every language  $l \in L$  is countable.

Proof: By definition,  $L \subseteq D$ . Therefore, for every language  $l \in L$  there must exist a  $d \in D$  where  $l = d$ . By Theorem 3.14, each  $l$  is also countable. Q.E.D.

Instead of working from the uncountably infinite set  $D$  (see Theorem 3.07), the size of  $L$  can be determined by working from the empty set.

Theorem 3.15: The set of formal languages over  $Z$  is not empty.

Proof: From set theory, the cardinality of  $\emptyset$  is zero but the cardinality of the set  $\{\emptyset\}$  is one. From Definition 3.19,  $\emptyset$  is a regular set. From Definition 3.21, a regular set must be accepted by a finite automaton. Applying Theorem 3.12,  $\emptyset$  is a type-3 language. Therefore, if all grammars generated only the empty set,  $L$  must still contain  $\emptyset$  as a language member. From Definition 3.08,  $D$  also contains  $\emptyset$  as a member. Hence,  $|L| \neq 0$ . Q.E.D.

Theorem 3.16: The cardinality of the set  $L$  of languages over  $Z$  is not finite.

Proof: Applying Theorems 3.06 and 3.10, the set of all finite subsets of  $Z^*$  is countably infinite. From Definition 3.19, every finite set is a regular set. Applying Definition 3.21 and Theorem 3.12, these regular sets are members of the type-3 languages over  $Z$ . From Chomsky's inclusion hierarchy (Theorem 1.01 of Section I), type-3 languages are a proper subset of type-0 languages. Since  $L$  contains all the type-0 languages by definition,  $L$  also contains these finite languages. Therefore, applying Definitions 3.09 and 3.10, the cardinality of  $L$  cannot be finite. Q.E.D.

Recognizing that the finite sets have been accounted for in the type-3 languages, a theorem and a corollary regarding the cardinality of each of the remaining sets of words and languages can be proven.

Theorem 3.17: For each  $d \in D$  and  $d \notin$  type-3 languages, then  $d$  has countably infinite cardinality.

Proof: By Theorem 3.14, each  $d$  is countable. Applying Definition 3.13, only the countably infinite sets remain. Q.E.D.

Corollary 3.02: For each language  $l \in L$  and  $l \notin$  type-3 languages, then  $l$  has countably infinite cardinality.

Proof: By definition,  $L \subseteq D$ . Therefore, for every language  $l \in L$  there must exist a  $d \in D$  where  $l = d$ . Applying Theorem 3.17,  $l$  has countably infinite cardinality. Q.E.D.

Even though  $L$  and  $D$  have been shown to contain a countably infinite subset of languages, conclusions regarding the cardinality of  $L$  cannot yet be reached. This is because Theorem 3.08 shows that a countably infinite subset exists in both countably infinite and uncountably infinite sets. Turing machines are needed to resolve this problem.

Theorem 3.18: The cardinality of the set of formal languages  $L$  over  $Z$  is countably infinite.

Proof: In the proof of Theorem 3.16,  $L$  was found to contain all the finite languages. Applying Theorem 3.10, these finite languages form a

countably infinite subset of  $L$ . Therefore,  $L$  must be at least countably infinite in size.

Now, from Definition 3.17 and Theorem 3.11, Turing machines accept type- $\emptyset$  languages. Depending on how the reject or loop criteria is interpreted (e.g. one machine could reject only one nontype- $\emptyset$  set while another could reject several different nontype- $\emptyset$  sets), it is possible for more than one Turing machine to accept a given type- $\emptyset$  language. However, Definition 3.17 states that at least one Turing machine exists for every type- $\emptyset$  language. Therefore, applying Theorem 3.13, there are at most a countably infinite number of Turing machines that could accept type- $\emptyset$  languages. Hence, there are at most a countably infinite number of type- $\emptyset$  languages.

Since  $L$  is bounded both from above and from below by countably infinite cardinality,  $L$  must be countably infinite in size. Q.E.D.

With  $L$  being countably infinite, proper inclusion of  $L$  in  $D$  can now be proven.

Theorem 3.19: The set  $L$  of formal languages over  $Z$ , is a proper subset of  $D$ .

Proof: From Theorem 3.07,  $D$  has cardinality  $c$  (uncountably infinite). On the other hand, Theorem 3.18 has shown  $L$  has cardinality  $\aleph_0$

(countably infinite). Recalling that  $L$  was defined such that it is a subset of  $D$  and applying Theorem 3.03, it follows that  $L$  must be a proper subset of  $D$ . Q.E.D

The proper inclusion of  $L$  in  $D$  supports the existence of nonrecursively enumerable sets. Formal language theorists have spent considerable effort in developing nonrecursively enumerable sets to prove the inclusion of  $L$  in  $D$ . One example of such a set is the countably infinite set representing the class of total recursive functions (92:562). However, theorists sometimes propose a nonrecursively enumerable set that is later proven to be recursively enumerable. Appendix C contains a proof that a nonrecursively enumerable set constructed by Manna is actually recursively enumerable (9:39).

Now that set of formal languages over a single alphabet has been proven to be included in  $D$  and has countably infinite cardinality, all that is left to do is to determine the size of the set of languages over all possible finite alphabets.

Theorem 3.20: The set of formal languages over all finite alphabets has countably infinite cardinality.

Proof: From AFL theory, all the finite alphabets for formal languages are obtained from a countably infinite alphabet (97:33). From Theorem 3.10, the set of possible finite alphabets has countably infinite cardinality. Since every

set has the empty set  $\phi$  as a finite subset, this countably infinite set of alphabets must contain  $\phi$ . However, by Definition 3.01, an alphabet cannot be the empty set. Therefore, the empty set must be removed from this group of alphabets. By applying the  $f(x)=x-1$  mapping on the indexes of the remaining finite sets, there is still a countably infinite number of finite alphabets remaining.

Now, since theorem 3.17 established that for a single, finite alphabet a countably infinite number of type-0 languages exist, let  $LL(i)$  represent the countably infinite set of type-0 languages for the  $i^{th}$  alphabet. Then, perform the set union operation over the  $LL(i)$  language sets. Since there are a countably infinite number of alphabets,  $i$  ranges over the natural numbers. Therefore, it follows from Definition 3.12 and Theorem 3.05 that the resulting set has countably infinite cardinality. Q.E.D.

With the size of the set all formal languages established, the formal language space analysis is complete. Because languages and grammars are related, an analysis of the size of grammar space should also be provided. Since the main results of this dissertation do not rely directly on theorems related to grammar space, the grammar-space



analysis is given in Appendix D for reference. Having established the size of formal language space, the investigation of the production rules and semantic networks can now begin.

#### IV. Representation Definitions

With the size of the language space established as countably infinite, the first step towards developing the formal representation languages is generating definitions for production rules and semantic networks.

##### Definition of Production Rules

The literature abounds with different descriptions of production rules (see references 5, 40, 41, 50, 51, 56, 66, 73, 74, 80 and Appendix B for examples). Trying to write a definition that would contain all the unique characteristics of each variation could result in numerous pages of confusing and possibly conflicting details. However, it was noted that these characteristics could be grouped into categories based upon the complexity of the knowledge that was to be represented. Each category could then be described by a statement that covered the member characteristics.

The results of this classification for the previously noted references are given in Table II and Table III. Because of the different natures of the "test only" antecedent and the "action" consequent of a rule, separate categorization of each was necessary. Additionally, the categories were ordered from less complex characteristics to more complex. Recalling that inclusion hierarchies are studied in formal language theory, the categories themselves were also established as such a hierarchy to support the

Table II. Categories of Antecedents of Production Rules

<u>Category</u>	<u>Characterization</u>
1	The antecedent contains a single string of symbols representing a single fact where existence of the fact in the database indicates that the fact is true.
2	The antecedent contains conjunctions of single facts [e.g. (Bill AND Cat)].
3	The antecedent contains conjunctions of total predicates operating on facts. [e.g. Red(ball) AND Owner(Joe, bat)].
4	The antecedent contains Boolean combinations (including nesting) of computationally primitive operations of matching and detecting.
5	The antecedent contains unit (?X in LISP) and segment (\$X in LISP) free-variables.
6	The antecedent can test values assigned to facts.
7	The antecedent can call complex functions embedded in the literals to determine the truth value.
8	The antecedent can alter the database of facts without firing any rules.
Note: The categories are specified in an inclusion hierarchy form: $8 \supset 7 \supset 6 \supset 5 \supset 4 \supset 3 \supset 2 \supset 1$	

language development goal. That is, the characteristics of a lower numbered category are properly included in those of a higher numbered category (see Figure 12). Therefore, as the category number increases, the number of characteristics and the complexity of the characteristics that a rule contains also increases. The top category level

Table III. Categories of Consequents of Production Rules

<u>Category</u>	<u>Characterization</u>
1	The consequent contains a single string of symbols representing a single fact which is added to the database as a true fact when the antecedent is true.
2	The consequent adds multiple facts to the database with use of the AND operator.
3	The consequent contains unit (?X in LISP) and segment (\$X in LISP) free-variables.
4	The consequent can perform substring replacement within a given fact.
5	The consequent can perform value assignment actions to facts.
6	The consequent can call procedures and other rules.
7	The consequent can manipulate the database and produce side effects which do not involve the rules.
Note: The categories are specified in an inclusion hierarchy form: 7 $\supset$ 6 $\supset$ 5 $\supset$ 4 $\supset$ 3 $\supset$ 2 $\supset$ 1	

indicated in Figure 12 represents category 8 for the antecedent hierarchy and category 7 for the consequent hierarchy.

While this classification effort condensed the set of characteristics, a definition which would cover all the possible combinations of antecedent categories with consequent categories would still be very complex. However, by following the techniques used by Feder (85), this complexity problem can be solved. Feder achieved success by studying a less complex language to determine some of the

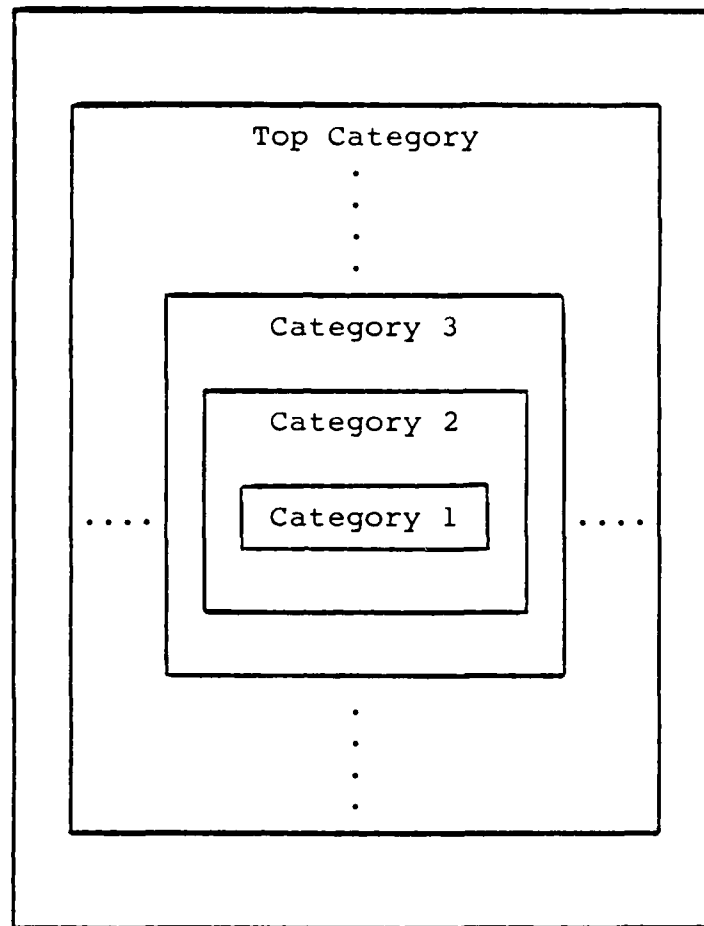


Figure 12. Inclusion Hierarchy

characteristics of more complex members of the same language hierarchy. Therefore, the solution is to utilize a similar approach and develop a definition of production rules based on the "category-1" characterizations of Tables II and III. This definition can then be used to generate a less complex language version.

Having selected the category level of the definition, the form that is used in the definition to represent the characteristics needs to be chosen. While generating the categories from the previous references, it was noted that

the most common form used in defining a production rule is a tuple form [e.g.  $a \rightarrow b$ ,  $(a,b)$ ]. Therefore, to maintain compatibility with the literature and formal language theory, a tuple form is chosen.

Now that the form selection is complete, the definition of production rules that have only category-1 characteristics is given. The actual number of rules contained within a representation is investigated in Section V.

Definition 4.01: For any nonempty, finite alphabet  $Z$ , a production rule that has only category-1 characteristics (see Tables II and III) is defined as a 3-tuple. The 3-tuple is of the form "amb" where "a" and "b" represent individual facts, "m" is a special symbol disjoint from  $Z$  which represents the implies operator,  $a, b \in Z^+$  and  $a \neq b$ . The form amb reads "a implies b."

In order to better understand the definition, the specified restrictions need to be explained. The concatenated word-form of the 3-tuple was selected in order to support the later development of a language version. By not using delimiters between the elements of the 3-tuple to show separation, later language development efforts would not have to be involved with blank spaces and commas within a production-rule word. However, by constraining  $m$  to be disjoint from  $Z$  and to be placed as the center element

AD-A172 516

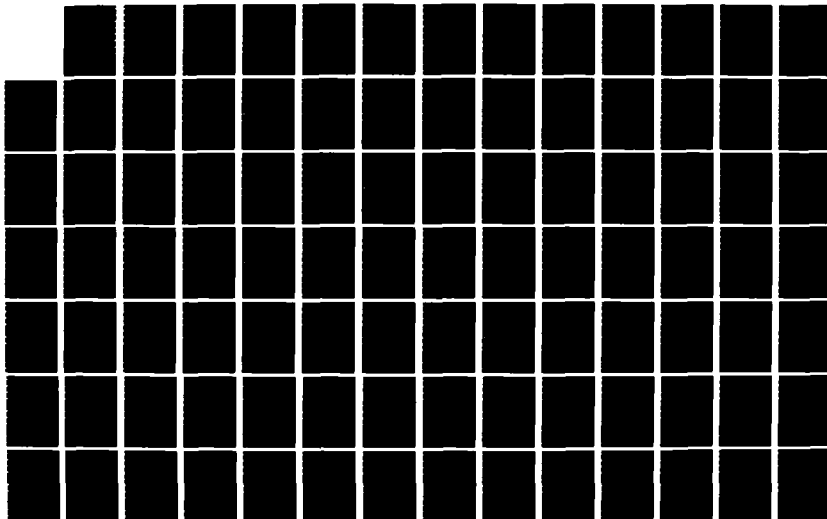
A NON-COGNITIVE FORMAL APPROACH TO KNOWLEDGE  
REPRESENTATION IN ARTIFICIAL (U) AIR FORCE INST OF  
TECH WRIGHT-PATTERSON AFB OH SCHOOL OF ENGI  
J A MCHANNAMA JUN 86 AFIT/DS/ENG/86J-1

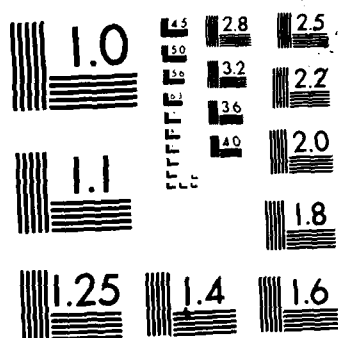
2/4

UNCLASSIFIED

F/G 9/4

NL





MICROCOPY RESOLUTION TEST CHART  
NATIONAL BUREAU OF STANDARDS 1963-A



within the 3-tuple,  $m$  can act as a separator as well as an operator if necessary.

Within this 3-tuple word-form, several restrictions involving the facts contained in the antecedent and consequent need explanation. First, single facts were not constrained to any limited set of words over  $Z$  so that the generality of the definition could be maintained. Because programmers arbitrarily assign symbol strings to represent facts, one may use "car" to represent a specific fact while another may use "auto" to represent the same fact. Therefore, the definition must provide the entire set of symbol strings for the programmers use.

Second, even though all words over  $Z$  were represented as facts, the empty word  $e$  was not added to the set because of its concatenation property:  $ea=a$  or  $ae=a$ . Since the 3-tuple was restricted to be a concatenated word-form, a grammar generating the 3-tuple word with an alphabet containing  $e$  could produce words such as "mb" or "am" instead of "emb" or "ame." These generated words are not 3-tuples and so are not production rules by definition.

Finally, when using any two of the previously constrained individual facts within a given rule, system operating efficiencies dictate that the facts be disjoint. Rules which have equal antecedent and consequent entries could cause a system to continuously fire the rule after its initial activation. Since a fact is true once it is placed in the database, adding multiple copies of the same fact to

the database does nothing to increase its truth value. Therefore, this continuous firing and updating action would cause unnecessary computational overhead during the system's operation.

#### Definition of Semantic Networks

Because semantic networks also have widely varying definitions (see references 5, 52, 53, 54, 62, 65, 68, 70, 71, 72, 73, 74, 78, 79 and Appendix B), the same methodology that was previously used for developing the production-rule definition can also be used to develop the semantic-network definition. Therefore, applying the same categorization approach, semantic networks can be characterized as shown in Tables IV, V and VI. Because of the differences between nodes and links, two separate hierarchies were constructed. Additionally, a hierarchy was developed to handle the combinations of links which do not depend on the characteristics of the links being combined.

Continuing to apply the same methodology, the semantic-network definition is based on the category-1 characterizations provided in the three associated tables. However, the form used in the definition is not based on the most common form as was the case for production rules. The most common form used to define a semantic network is a directed graph which has labeled nodes and links. Another form used is a tuple form which represents the relationship between only two nodes [e.g. (node-1, relation, node-2)]. Recalling that

Table IV. Categories of Nodes of Semantic Networks.

<u>Category</u>	<u>Characterization</u>
1	A node contains a single string of symbols representing objects, categories, concepts, sets, etc., but does not contain relations.
2	A node contains a single string of symbols representing relations, predicates, attributes or values.
3	A node contains unit (?X in LISP) and segment (\$X in LISP) free-variables in support of quantification representation.
4	A node contains functions, procedure calls, rules or special symbols.
5	A node contains prototypes, depictions, roles or descriptors.

Note: The categories are specified in an inclusion hierarchy form:  $5 \supset 4 \supset 3 \supset 2 \supset 1$

the goal of this dissertation is to compare the production-rule and semantic-network languages, the task can be made easier if the words in the languages are of similar form. Therefore, the tuple form is chosen for the semantic-network definition.

With this groundwork completed, the definition of semantic networks that have only category-1 characteristics is given. As was the case for production rules, the size of the semantic network is investigated in Section VI.

Table V. Categories of Individual Links of Semantic Networks

<u>Category</u>	<u>Characterization</u>
1	A link represents a binary relation between two nodes where the relation can be thought of as a predicate of the form $\text{Relation}(\text{tail\_node}, \text{head\_node})$ .
2	A link represents a pointer without any relation such as an attribute-value pointer or a pointer associated with a relation-node.
3	A link represents an n-ary relation.
4	A link represents exception cases.
Note: The categories are specified in an inclusion hierarchy form: $4 \supset 3 \supset 2 \supset 1$	

Table VI. Categories of Link-Combinations of Semantic Networks

<u>Category</u>	<u>Characterization</u>
1	The links are treated as stand-alone independent entities.
2	The links are combined by disjunction, conjunction and negation.
3	The links are combined to represent the antecedent and consequent of implications.
Note: The categories are specified in an inclusion hierarchy form: $3 \supset 2 \supset 1$	

Definition 4.02: For any two disjoint, nonempty, finite alphabets,  $Y$  and  $Z$ , a semantic network that has only category-1 characteristics (see Tables IV, V and VI) is defined as a collection of 3-tuples. Each 3-tuple is of the form "arb" where "a" and "b" represent the node labels, "r" represents the relation between the nodes,  $a, b \in Z^+$ ,  $r \in Y^+$  and  $a \neq b$ . The form arb reads "a is related to b by r." That is, the link arrow goes from a to b.

As was the case for production rules, the semantic-network definition can also be better understood if the restrictions are explained. Except for the naming conventions used, the explanations for the restrictions of the 3-tuple to a concatenated word-form, the placement of r, the use of all the words over  $Z$  for node labels, the use of all words over  $Y$  for the relations and the elimination of the empty word are identical to those given for similar restrictions in the definition of a production rule. Therefore, they will not be repeated here. The reader is referred to the paragraphs following Definition 4.01 for these explanations. However, there are two additional restrictions that need explanation.

The first is the restriction that the node labels and relations be constructed from disjoint alphabets. This restriction comes directly from the category-1 characteris-

tics given in Table IV. There, a node was restricted not to contain a relation. By using disjoint alphabets, nodes can be guaranteed not to contain any words that are symbolically identical to the relation-words.

The second and final restriction requiring explanation is the computational efficiency of  $a \neq b$  within any given 3-tuple. Recall that the inference mechanism of spreading activation (see Section I) relies on the capability to follow links throughout the network. If a node was encountered that was related directly to itself, the activation procedure could loop forever on that path. Therefore, if nodes of this type were allowed, additional computational overhead would have to be expended in order to prevent the activation procedure from entering these loops.

#### Related Definitions

The definitions of production rules and semantic networks are not the only definitions used herein. According to Definitions 4.01 and 4.02, a production rule requires a set of facts and a semantic network requires a set of node labels and a set of relations. Because the terms "knowledge-words" and "knowledge-languages" are used later when referring to these sets, separate definitions of these terms are needed for clarity.

Definition 4.03: For a production rule, a knowledge-word is a single fact-word contained in the set

of facts and a knowledge-language is the entire set of fact-words.

Definition 4.04: For a semantic network, a knowledge-word is either a single node-label word or a single relation-word and the knowledge-language is the union of the node-label and relation sets.

Having generated the necessary definitions of production rules and semantic networks, the formal language equivalents of these two knowledge representations is developed.

## V. Production-Rule Language and Characteristics

Given these category-1 definitions for production rules and semantic networks, the formal language and characteristics for each of these representations can now be developed. Because of the extensive effort required to determine language equivalents, each representation is covered in its own section. Production rules are covered first.

### Production-Rule Language

There are two different approaches that are used to determine the language membership of this production-rule representation. The first approach is to investigate a set of the finite, yet variable length, strings  $amb$  generated by concatenation of the symbols from the alphabet  $Z \cup \{m\}$ . In this case, the characteristics (e.g. word membership in the set of facts, individual consideration of symbols making up a fact-string, language-type of the fact-set) of the individual fact-strings,  $a$  and  $b$ , are evaluated along with the representation's  $amb$  strings over  $Z \cup \{m\}$ . In other words, the representation's structure is not considered to be separated from the facts it contains.

The second approach that will be applied is to use the abstract 3-tuple form of the production-rule strings and consider each of these strings to be of a fixed length, three. Here, each individual fact is considered as a single symbol in the language-alphabet. Individual characteristics



of facts are hidden by the abstract nature of this knowledge representation's structure. Only the size of the fact-set is considered.

Case I: Facts are Words. The best place to start the production-rule language development is at the individual word level where consideration is given to the characteristics of the fact-set.

Theorem 5.01: Each production rule  $amb$  constructed under the constraints of Definition 4.01 of Section IV is a finite word over  $Z \cup \{m\}$ .

Proof: From Definitions 3.01 and 3.02 of Section III, a word over any nonempty, finite alphabet is finite in length. From Definition 4.01,  $Z$  was defined to be nonempty and finite and  $m$  was defined to be a single symbol. From set theory, the union of two finite sets remains finite and so the new alphabet,  $Z \cup \{m\}$ , is also nonempty and finite.

In addition to this finite alphabet, each  $a$  and  $b$  are also finite in length (see Definitions 3.03 and 3.04 of Section III). Since the sum of finite numbers remains finite, the length of  $amb$  is finite. Therefore, each  $amb$  over  $Z \cup \{m\}$  meets the requirements for being a finite word. Q.E.D.

Using these individual rule-words, let "PR" represent the set consisting of all these words for a given  $Z$ .

	C0	C1	C2	C3	.....
C0m	XX	C0mC1	C0mC2	C0mC3	.....
C1m	C1mC0	XX	C1mC2	C1mC3	.....
C2m	C2mC0	C2mC1	XX	C2mC3	.....
C3m	C3mC0	C3mC1	C3mC2	XX	.....
.	.....	.....	.....	.....	.....
.	.....	.....	.....	.....	.....
.	.....	.....	.....	.....	.....

Figure 13. PR Word Matrix

That is, PR contains all possible 3-tuples over  $Z \cup \{m\}$  meeting Definition 4.01 of Section IV. The size characteristic of this word-set can be determined from the following theorem.

Theorem 5.02: The set PR consists of a countably infinite number of words.

Proof: From Theorem 3.06 of Section III, the set  $Z^*$  is countably infinite. Recognizing that  $Z^+ = Z^* - \{e\}$  and using the mapping  $f(x) = x - 1$  on the natural numbers, it follows that the set  $Z^+$  is also countably infinite.

Now, construct the matrix equivalent of the PR set as shown in Figure 13. The symbols  $C0, C1, \dots$  represent the words contained in  $Z^+$  and the symbol m is the implies relationship.

The symbol "XX" represents those words which are not contained in PR due to the  $a \neq b$  constraint.

For this matrix, it follows from the size of  $Z^+$  that there are countably infinite rows. Because there is exactly one word per row where  $a=b$ , a set consisting of all the XX words must also be countably infinite. Using the mapping of  $f(x)=x-1$  once again, each row is found to still contain a countably infinite number of entries when the  $a=b$  word is removed. Since PR is equivalent to the union of the rows of this matrix with the XXs removed, it follows from Theorem 3.05 of Section III, that PR consists of a countably infinite number of words. Q.E.D.

Now that PR has been shown to contain a countably infinite number of formal words, the next step is to classify PR as a formal language. Recalling that formal language theorists have shown the existence of countably infinite hierarchies of languages within a given language-type (Chomsky) as well as across types (AFLs), trying to establish that PR resides in one of the known hierarchies could take a considerable number of "test and evaluate" actions. Because of the existence of machines (e.g. Turing, finite automata) that accept only the languages of a given type (93), the set PR is only evaluated against the four-

level Chomsky hierarchy (8). Formal language theorists have also developed a significant amount of characteristics about languages residing in this finite hierarchy.

To determine the language placement of PR in this finite hierarchy, the impact on the language-type of PR caused by the constraints on the facts (see Definition 4.01 of Section IV) is investigated. First, all of the constraints are relaxed and the language-type of a set of production rules is determined. Then, the constraints are applied one at a time and the language-type is re-evaluated. To begin, the fact-set is allowed to contain the empty word  $\epsilon$  and in a given amb string  $a$  is allowed to equal  $b$ .

Theorem 5.03: The set of all production-rule words, each word of the form  $amb$ , where  $Z$  is a nonempty finite alphabet,  $a, b \in Z^*$  and  $m$  is a single symbol disjoint from  $Z$ , is a type-3 language.

Proof: With all of the previous restrictions on the set of facts eliminated, the set of production-rule words can be represented as the concatenation of the words of three sets:  $Z^*\{m\}Z^*$ . Because  $Z$  is a finite alphabet and  $m$  is a single symbol, it follows from Definition 3.19 of Section III that  $Z$  and  $\{m\}$  are regular sets and so is  $Z^*$ . Applying this regular set definition again,  $Z^*\{m\}Z^*$  is found to be a regular set. Finally, applying Definition 3.21 and Theorem 3.12 of Section III to this

concatenated set, this set of production-rule words is found to be a type-3 language.

Q.E.D.

Now that the unrestricted case has been proven to be a type-3 language, the empty word constraint is applied and the language-type is re-evaluated.

Theorem 5.04: The set of all production-rule words, each word of the form  $amb$ , where  $Z$  is a nonempty, finite alphabet,  $a, b \in Z^+$  and  $m$  is a single symbol disjoint from  $Z$ , is a type-3 language.

Proof: In this case, the set of production-rule words can be represented by the concatenation of three sets:  $Z^+\{m\}Z^+$ . In reference 94, pages 30-31,  $Z^+$  is proven to be a regular set. Therefore, applying Definition 3.19 of Section III,  $Z^+\{m\}Z^+$  is a regular set. From Definition 3.21 and Theorem 3.12 of Section III, this set of production-rule words is a type-3 language.

Q.E.D.

From this theorem, the removal of the empty word from the set of facts is seen not to affect the language-type of the set of production rules. However, this is not the case for the  $a \neq b$  constraint as is seen in the next two theorems.

Theorem 5.05: The set of all production-rule words,  $PR$ , where each word  $amb$  meets the requirements of Definition 4.01 of Section IV, is not a type-3 language.

Proof: The PR set is identical to one of the "center-marker" languages studied by Haines (98):

$CM = \{xcy \mid x, y \in R, R \text{ is a regular set over } Z, x \neq y, c \notin Z\}$ . However, Haines did not prove directly that some CM languages are not type-3. Therefore, the PR case will have to be proven in detail.

For this proof assume that PR is a regular set. From the regular set closure properties (92:186), if PR is a regular set then so is its complement;  $CR = [Z \cup \{m\}]^* - PR$ . If the complement CR is a regular set, then so is its intersection with the regular set of Theorem 5.03;  $IR = CR \cap \{amb \mid a, b \in Z^*, m \notin Z\}$ .

Now, this last intersection yields  $IR = \{xmx \mid x \in Z^*\}$ . By applying the iteration theorem of regular sets to IR (96:47), if IR is a regular set, it must contain words of the form "xmy" where  $|x| \neq |y|$ . But, IR contains only words where  $|x| = |y|$ . Hence, IR cannot be a regular set.

The fact that IR is not a regular set indicates that PR is not a regular set. This is determined by following the steps used to construct IR in reverse. Since IR is not a regular set, then CR cannot be a regular set. Since CR is not a regular set, then PR cannot

be a regular set. Hence, PR is not a type-3 language. Q.E.D.

Because of the relationship of PR to one of the centermarker-languages of Haines, the actual language-type of PR has already been determined by him.

Theorem 5.06: The set of all production-rule words, PR, where each word  $amb$  meets the requirements of Definition 4.01 of Section IV, is a type-2 language.

Proof: The PR set is identical to one of the centermarker-languages studied by Haines (98):  $CM = \{xcy \mid x, y \in R, R \text{ is a regular set over } Z, x \neq y, c \notin Z\}$ . Haines proved that the CM languages are at least type 2. The details of his proof will not be repeated here. By applying Theorem 5.05, PR is found to be exactly a type-2 language. Q.E.D.

So, PR has been proven to be a type-2, context-free, language. By considering the facts as a language themselves, the production-rule language has been shown to be dependent on the knowledge-language (see Definition 4.03 of Section IV) as well as the rule's structure. More is discussed about the characteristics of PR later. But first, consider the case where the facts are defined to be symbols in an alphabet.

Case II: Facts are Symbols. In order to "hide" the language characteristics of facts in the production-rule

strings, the facts will need to be treated as arbitrary entities. To achieve this, each fact is considered to be a single alphabet symbol of unit length. Because only the individual characteristics of the facts are to be hidden, this alphabet of facts remains countably infinite in size. Otherwise, the set does not contain all the possible facts in  $Z^+$ . The definition of the production-rule word has to be modified as follows to incorporate these arbitrary facts:

**Definition 5.01:** For an arbitrary, countably infinite set of facts  $F$ , a production rule that has only category-1 characteristics (see Tables II and III) is defined as a 3-tuple. The 3-tuple is of the form "amb" where "a" and "b" represent individual facts, "m" is a special symbol disjoint from  $F$  which represents the implies operator,  $a, b \in F$ ,  $|a|=|b|=1$  and  $a \neq b$ . The form amb reads "a implies b".

With this definition in hand, let "PA" represent the set consisting of all the 3-tuples meeting Definition 5.01. For this set, the first step is to determine if each 3-tuple is a formal word.

**Theorem 5.07:** Each production rule of the form amb meeting Definition 5.01 is not a finite word.

**Proof:** This follows directly from the definition of a word. While the length of amb is three and hence finite, the overall alphabet,  $F \cup \{m\}$ ,



that the strings are constructed from is countably infinite. From Definitions 3.01 and 3.02 of Section III, words are constructed from only finite alphabets. Therefore, the amb strings are not finite words. Q.E.D.

This theorem makes a very strong statement at the very beginning of the investigation of PA. The amb strings were purposely made finite in length to make these strings meet the finite length requirement of formal words (see Definition 3.02 of Section III). Yet, the countably infinite alphabet prevents the strings from being formal words. The size of this alphabet also leads to the following theorem regarding the PA set.

Theorem 5.08: The set of production rules PA consisting of all possible strings amb meeting the Definition 5.01 is not a formal language.

**Proof:** Because languages require finite words as members, PA can be proven not to be a formal language immediately by applying Theorem 5.07. However, to emphasize the critical nature of the size of the alphabet, an alternate proof is given.

Recall from Definition 3.07 of Section III that a language is the set of all words that can be generated by a grammar. From the definition of a grammar (Definition 1.02 of Section I), a grammar has to have a finite set

T of terminal symbols. The overall alphabet,  $F \cup \{m\}$ , is this terminal set T for the PA grammar. But, this alphabet is countably infinite. Therefore, a grammar does not exist that generates all the finite strings (words) in the PA set; thus PA cannot be a type-0 formal language. Q.E.D.

Besides establishing that PA is not a language, Theorem 5.08 emphasizes that for strings of symbols, finite length is a necessary but not sufficient condition to make the strings formal words. This is but one of the characteristics that have to be taken into account when attempting to map the knowledge representations into formal language structures. There are other characteristics that impact the PR and PA sets as well.

#### Production-Rule Characteristics

Even though PA is not a formal language (Theorem 5.08), the characteristics of this set reveal several important facts that affect the class of languages that contains the PR language (see Theorem 5.06). Therefore, the characteristics of PA are covered first. Then, the characteristics of the production-rule set PR are developed.

PA Characteristics. To investigate the PA set further, the number of strings contained in PA is determined.

Theorem 5.09: The set PA consisting of all possible 3-tuples over F of the form amb meeting Definition 5.01 is countably infinite.

Proof: Because the alphabet F is countably infinite, the proof of Theorem 5.02 can be applied directly to this theorem. That is, let the  $C_0, C_1, \dots$  be members of F instead of  $Z^+$ . Therefore, it follows that PA is countably infinite. Also, the subset of words where  $a=b$  is countably infinite for the PA case as well. Q.E.D.

It is important not to try to prove that PA is countably infinite by applying Theorem 3.17 of Section III. Theorem 3.17 requires, but does not so state, that the strings be formal words. From Theorem 5.07, PA does not contain these required formal words. But, PA can be forced to contain formal words by restricting the set F to be finite.

Theorem 5.10: For an arbitrary, nonempty, finite subset of F, FS, all of the production rules of the form amb that meet Definition 5.01 over FS instead of F are finite words.

Proof: Because FS is nonempty and finite, the overall alphabet for the production rules,  $FS \cup \{m\}$ , is also nonempty and finite. Therefore, these production-rule strings meet the first of the two constraints for formal words (see Definition 3.01 of Section III).

The second constraint, finite length (see Definition 3.02 of Section III), is met by the constraint that  $|a|=|b|=|m|=1$ . Hence, each amb string is of length three, a finite number. Therefore, each amb string is a finite word over  $FS \cup \{m\}$ . Q.E.D.

Since production-rule words exist for this special case, the search for a language-type can begin. But this search is very short in comparison to the previous search for the PR language.

Theorem 5.11: The set of production rules PF consisting of all possible strings meeting Definition 5.01, except that the facts are selected from an arbitrary, finite subset FS of F, is a finite set and hence a type-3 language.

Proof: Let the size of FS be a nonzero, finite number, M. Following the same technique of matrix construction used in Theorem 5.02, construct the matrix equivalent of PF as shown in Figure 14. The symbols  $C_0, C_1, \dots, C_M$  represent the a and b fact-symbols in FS. The symbol XX represents the  $a=b$  cases. The symbol m is the implies relationship.

From Figure 14, the size of the matrix is  $M^2$ . Because there is only one XX entry per row, the overall size of PF is  $M(M-1)$ . Since M is finite,  $M(M-1)$  is also finite.

	C0	C1	C2	...	CM
C0m	XX	C0mC1	C0mC2	...	C0mCM
C1m	C1mC0	XX	C1mC2	...	C1mCM
C2m	C2mC0	C2mC1	XX	...	C2mCM
...	...	...	...	...	...
CMm	CMmC0	CMmC1	CMmC2	...	XX

Figure 14. PF Word Matrix

Therefore, PF consists of a finite number of words.

Using PF's finite size, the fact that the amb strings are words and applying Definitions 3.19, 3.21 and Theorem 3.12 of Section III, PF is found to be a type-3 language. Q.E.D.

Theorem 5.11 only dealt with a single arbitrary case of converting a subset of strings in PA into some formal language. The next theorem expands this into a class of languages.

Theorem 5.12: There exists a countably infinite number of finite production-rule languages where the nonempty fact-sets are selected from an arbitrary, countably infinite set of symbols.

Proof: Theorem 5.11 was based on selecting a finite subset of facts from F. Applying Theorem 3.10 of Section III, there is a countably infinite

number of finite subsets in  $F$ . Because the empty set  $\emptyset$  is always a finite subset of any set, the nonempty constraint requires that it be removed from this group of subsets. After removing the empty set and applying the  $f(x)=x-1$  mapping on the indexes of the remaining subsets, there is still a countably infinite number of nonempty finite subsets available for use in generating finite production-rule languages. Hence, there exists a countably infinite number of finite production-rule languages. Q.E.D.

Because finite languages have been studied extensively by Maurer et al. (99; 100), further investigation of the characteristics of these finite production-rule languages is not accomplished. However, it is important to notice that when the facts are arbitrary, countably infinite, symbols, no single production-rule language exists that can contain all possible rules. See Section IX for discussions on additional affects that the knowledge-languages (see Definition 4.03 of Section IV) have on production rules.

Category 1 Class. Since the PR version of production rules does contain all possible production-rule words for a given  $Z$ , the characteristics of the PR language merit further investigation. Due to the fact that the general characteristics of context-free languages have been studied extensively, the "common" characteristics of type-2 lan-

guages (e.g. normal grammar forms, word membership algorithm, parsing algorithm) of PR alone are not investigated further. The reader is referred to references 6, 92, 93, 94, 95, 96 and the references in Appendix A for information on how to determine these characteristics.

However, there are characteristics of the class of languages which contains the PR language that directly impact later findings. Therefore, the characteristics of this class are covered in depth.

Theorem 5.13: Any set of production-rule words consisting of all the words meeting Definition 4.01 of Section IV, except that the set of facts for each rule-set is any nonempty regular set over  $Z$ , is at least a type-2 language.

Proof: This proof follows directly from Haines (98) and Theorems 5.06 and 5.12. Haines proved that for the centermarker-languages, the  $x$  and  $y$  strings of the centermarker-word  $xcy$  can be selected from any regular set and the resulting language is type 2. Here, the equivalent  $x$  and  $y$  are selected from any nonempty regular set over the fact-alphabet  $Z$ . Therefore, each of the production-rule languages in this theorem is upper bounded at a type-2 language.

Now, to show the "at least" criterion, the results of Theorems 5.06 and 5.12 are needed.

Theorem 5.06 established that the production-rule set PR is exactly a type-2 language. Since PR uses a nonempty regular set  $Z^+$  for the fact-set, PR meets the requirements of this theorem. Hence, a type-2 language exists that meets the constraints of this theorem.

Next, Theorem 5.12 established the existence of a countably infinite set of type-3 production-rule languages where the facts are selected from nonempty, finite, regular sets. If  $Z^+$  is used as the arbitrary, countably infinite set for Theorem 5.12, then these finite language are also contained in the set of languages covered by this theorem.

Therefore, since both type-2 and type-3 production-rule languages are contained in the set of languages covered by this theorem, it follows from Chomsky's language-hierarchy (8) and the type-2 upper bound that these languages are at least type 2. Q.E.D.

As was shown in Theorem 5.13, the production-rule language PR is only one of a number of possible production-rule languages each of which has only category-1 characteristics (see Tables II and III of Section IV). The size of this class of languages provides another important characteristic.



Theorem 5.14: The production-rule language PR consisting of all words meeting Definition 4.01 of Section IV is a member of a countably infinite class of type-2 production-rule languages.

Proof: This proof follows from Theorem 3.18 of Section III and Theorems 5.02, 5.06, 5.12 and 5.13. Theorem 5.13 established that type-2 production-rule languages can be constructed provided the facts are selected from a non-empty regular set. Theorems 5.02 and 5.06 revealed the existence of at least one language containing a countably infinite number of words, PR, that meets the constraints of Theorem 5.13. By using  $Z^+$  as the arbitrary countably infinite set, Theorem 5.12 shows the existence of a countably infinite set of finite production-rule languages that also meet the regular set constraint of Theorem 5.13. Since PR contains a countably infinite number of words, PR is not a member of this countably infinite set of finite languages. Hence, from set theory, the union of a finite set (the single PR language) and a countably infinite set (the finite languages) is countably infinite. So, PR is a member of a set of languages that is at least countably infinite in size.

To prove the upper bound on this class, the size of language space is needed. Theorem 3.18 of Section III established that the size of language space for a given alphabet is countably infinite. Therefore, the upper bound for this class of production-rule languages is also countably infinite. Since both the upper and lower bounds agree, this class of production-rule languages is countably infinite in size. Q.E.D.

So, at the category-1 level of the definition-hierarchy (see Figure 12 in Section IV) there exist countably infinite numbers of production-rule languages (both finite and countably infinite in size) of which PR is a member. The fact that this class consists of all possible production-rule languages over regular fact-sets yields another characteristic of this class of languages.

Theorem 5.15: The class of category-1 languages consisting of production-rule languages where each language consists of all the words meeting Definition 4.01 of Section IV, except that the set of facts for each language is any nonempty regular set over  $Z$ , is not closed under the union operation.

Proof: This proof follows from construction of a counter-example. For this proof, arbitrarily select the  $Z$  alphabet as follows:  $Z = \{c, d, g,$

h, k}. For this alphabet, arbitrarily select two finite, regular sets as follows:  $S_1 = \{c, d\}$  and  $S_2 = \{g, h\}$ . Now, construct the two production-rule languages for these sets:  $L_1 = \{cmd, dmc\}$  and  $L_2 = \{gmh, hmg\}$ . Finally, using these two languages, form a third language by the union operation:  $L_3 = \{cmd, dmc, gmh, hmg\}$ .

By this construction,  $L_1$  and  $L_2$  meet the constraints needed to be members of the specified class of category-1 languages. That is, each of these two languages consists of all the words meeting Definition 4.01 of Section IV except that the facts are selected from some nonempty regular set over  $Z$ . However, the language formed from their union,  $L_3$ , does not meet these constraints.

The language  $L_3$  consists of production-rule words formed from a regular set of four individual facts:  $S_3 = \{c, d, g, h\}$ . But,  $L_3$  does not contain all possible production-rule words over  $S_3$ . For example, the legal word "cmg" is not contained in  $L_3$ . Yet, by the definition of the production-rule languages contained in this class, this specific word (and others) must be contained in the production-rule language over the  $S_3$  set of facts. Therefore,

L3 is a finite language but is not a production-rule language.

Since the union closure property requires that the third language L3 also be in the class, it follows that this class of category-1 production-rule languages is not closed under the union operation. Q.E.D.

This union closure property is not the only regular operation that this class of production-rule languages does not meet. This class also does not meet the product closure property. The product of two languages is the concatenation of a word from the first language to that of a word from the second language in order to form the words contained in a third language. To be closed under this operation, the language formed by the product operation must also be in the same class as the two original languages.

Theorem 5.16: The class of category-1 languages consisting of production-rule languages where each language consists of all of the words meeting Definition 4.01 of Section IV, except that the set of facts for each language is any nonempty regular set over Z, is not closed under the product operation.

Proof: This proof follows from the construction of a counter-example. For any two production-rule languages in this class of category-1 languages, the product operation generates

strings of the form "AlmBlA2mB2." Because a production-rule word has only one implies symbol  $m$ , the product operation generates illegal rule-words. Hence, the strings generated by the product operation cannot be contained in any production-rule language. So, this class of category-1 production-rule languages is not closed under the product operation. Q.E.D.

These closure properties of this class of production-rule languages lead to another characteristic of this language class. This characteristic is that this class of production-rule languages does not form an Abstract Family of Languages (AFL) (24).

Theorem 5.17: The class of category-1 languages consisting of production-rule languages where each language consists of all words meeting Definition 4.01 of Section IV, except that the set of facts for each language is any nonempty regular set over  $Z$ , is not an AFL.

Proof: From reference 24, an AFL contains languages that are closed under six operations: union, product, Kleene closure,  $\epsilon$ -free homomorphism, inverse homomorphism and intersection with a regular set. From Theorems 5.15 and 5.16, this class of production-rule languages is not closed under either union or product.

Therefore, this class of category-1  
production-rule languages cannot be an AFL.  
Q.E.D.

The result of Theorem 5.17 presents a major stumbling block to the further study of this class of production-rule languages. Because this class is not an AFL, the determination of additional characteristics of the class has to involve the analysis of the entire set of languages in the class. Thus, the multitude of characteristics already proven about AFLs by formal language theorists cannot be used to help expand the characteristic-set of this class of languages.

The characteristics of AFLs are not the only characteristics that do not apply to this class of languages. The characteristics of the general class of type-2 languages also do not apply. For example, the entire set of Chomsky's type-2 languages form an AFL but this class of category-1 production-rule languages does not. Another example is that the entire set of Chomsky's type-2 languages is closed under the regular operations of union and product (6:12), yet this class of languages is not.

So, while this class of category-1 production-rule languages is a set of type-2 languages, characteristics of the general type-2 languages cannot be used, without proof, when referring to this class of production-rule languages. Even though other characteristics of this class of production-rule languages as well as characteristics of

other classes of category-1 production-rule languages could be investigated, sufficient characteristics of production-rule languages have been identified for use in later investigations. The time has come to develop the semantic-network language equivalent.

## VI. Semantic-Network Language and Characteristics

Since the 3-tuple form of a production-rule representation has been successfully evaluated via formal language theory, the investigation can now turn to the evaluation of the 3-tuple form of a semantic-network representation. Because both representations use a 3-tuple form, some theorems about semantic networks can be proven by arbitrarily mapping the symbol  $m$  to a single relation-word in the relation-set. To make extensive use of this mapping, the semantic-network development parallels that of the production-rule effort.

### Semantic-Network Language

Following the production-rule methodology, there are two approaches that are used to determine a semantic-network language equivalent: node labels and relations are words; node labels and relations are symbols. However, for a semantic network there are two additional approaches that have to be considered: node labels are words and relations are symbols and vice versa. Because these last two approaches depend on the results of the "node label and relations are symbols" approach, they are addressed under that approach.

Case I: Word Approach. As was the case for the production-rule effort, the semantic-network development begins with analysis at the individual word level.



Theorem 6.01: Each set of semantic-network strings constructed under the constraints of Definition 4.02 of Section IV contains finite words of the form arb.

Proof: From Definitions 3.01 and 3.02 of Section III, a word over any nonempty, finite alphabet is finite in length. From Definition 4.02, the alphabets for the node-label set,  $Z$ , and the relation-set,  $Y$ , are each nonempty and finite. From set theory, the union of two finite sets remain finite. Therefore, the alphabet over which each arb string is constructed,  $Z \cup Y$ , is nonempty and finite.

Each of the  $a$ ,  $r$  and  $b$  components of the overall string is finite in length (see Definitions 3.03 and 3.04 of Section III). Since the sum of finite numbers remains finite, the length of arb is a finite number. Therefore, each 3-tuple, arb, over  $Z \cup Y$  is a finite word. Q.E.D.

Now that the individual 3-tuples are known to be finite words, the set "SN" can be constructed which contains all the possible words over  $Z \cup Y$  that meet Definition 4.02 of Section IV. The size of this set can be determined by expanding the proof given for the size of the PR production-rule language (see Theorem 5.02 of Section V).

Theorem 6.02: The set SN contains a countably infinite number of words.

Proof: The proof of this theorem follows directly from Theorem 5.02 of Section V and Theorem 3.05 of Section III. For Theorem 5.02, Figure 12 was constructed for the case of words of the form  $amb$  where  $m$  was a single symbol and  $a, b \in Z^+$ . In that case, there were found to be a countably infinite subset of cases for  $a=b$  as well as countably infinite rows. Each row was also found to contain countably infinite entries. So, the entire set PR was found to be countably infinite by application of the Theorem 3.05: the union of a countable set of countable sets remains countable.

Using this production-rule case, discard the "implies" meaning attached to  $m$  and rename the symbol  $m$  to be a single relation-word " $R_1$ ," where  $R_1 \in Y^+$ . Then, let the fact-alphabet  $Z$  be equated to the node-label alphabet. Under this mapping, the proof of Theorem 5.02 shows that there is a countably infinite number of words of the form " $aR_1b$ " in SN and a countably infinite number of  $a=b$  cases for a given  $R_1$ .

Now let  $R_1$  range over  $Y^+$ . Since  $Y^+$  is countably infinite (see Theorem 5.02 for the

proof of the size of  $Z^+$ ), there is a countably infinite number of subsets of words "aR(i)b" in SN, each containing countably infinite words. By applying Theorem 3.05, SN is found to contain a countably infinite number of words. Additionally, it follows that the union of all the countably infinite subsets where  $a=b$  is countably infinite. Q.E.D.

Because the set SN contains a countably infinite number of finite words, additional effort will be required to determine if it is a formal language. That is, since SN is not finite (a type-3 language), additional analysis is required to determine the language-type (if any) of SN. The semantic-network development follows the production-rule approach of using Chomsky's finite language-hierarchy to classify the languages (8). Additionally, the semantic-network language is developed by first removing all the constraints of Definition 4.02 of Section IV on the node labels and relations. Then, each constraint is applied, one at a time, so that the affects of these constraints can be identified.

Theorem 6.03: The semantic network that contains all of the words of the form arb, where Z and Y are nonempty, disjoint, finite alphabets and  $a, b \in Z^*$ ,  $r \in Y^*$ , is a type-3 language.

Proof: The set of semantic-network words for this case can be represented as the concatenation

of three sets:  $Z^*Y^*Z^*$ . Because  $Y$  and  $Z$  are finite, they are regular sets by Definition 3.19 of Section III. Applying Definition 3.19 once more,  $Z^*$  and  $Y^*$  are found to be regular sets as well. Then, applying the concatenation rule of Definition 3.19,  $Z^*Y^*Z^*$  is found to be a regular set. Applying Definition 3.21 and Theorem 3.12 of Section III, this regular set is proven to be a type-3 language. Q.F.D.

Now that the unrestricted case has been found to be a type-3 language, let the constraint be applied that the empty word,  $e$ , cannot be contained in either the node-label or relation set and evaluate again.

Theorem 6.04: The semantic network containing all of the words of the form  $arb$ , where  $Z$  and  $Y$  are nonempty, disjoint, finite alphabets and  $a, b \in Z^+$ ,  $r \in Y^+$ , is a type-3 language.

Proof: In this case, the set of semantic-network words can be represented as the concatenation of three sets:  $Z^+Y^+Z^+$ . Because  $Z$  and  $Y$  are finite alphabets,  $Z^+$  and  $Y^+$  are regular sets (94:30-31). Applying the concatenation rule of Definition 3.19 of Section III,  $Z^+Y^+Z^+$  is a regular set. Applying Definition 3.21 and Theorem 3.12 of Section III, this regular set is proven to be a type-3 language. Q.E.D.

The proof technique of Theorem 6.04 also applies to the cases where the empty word is allowed in one set (node labels or relations) but not the other. This conclusion follows from the fact that  $Z^*$ ,  $Z^+$ ,  $Y^*$  and  $Y^+$  are all regular sets. The concatenation process on any mix of these sets still yields a regular set. Because the case proven in Theorem 6.04 already applies directly to the constraints of the semantic-network definition, these other mixed cases are not established as formal theorems.

Even though this empty word constraint does not cause the language-type to change, the  $a \neq b$  constraint on the node labels affects the type-3 language finding. The following theorem relies heavily on the results found for the production-rule language.

Theorem 6.05: The semantic network, SN, consisting of all the words arb which meet Definition 4.02 of Section IV is not a type-3 language.

Proof: The proof will be by construction. First, assume that SN is a type-3 language. Then, from Definition 3.21 and Theorem 3.12 of Section III, all of the words in SN must be accepted by a finite automaton. Because the centermarker relation-set is a countably infinite regular set, this finite automaton must contain a finite number of states with looping paths to accept these relation-words. Since these relation-words are over a disjoint

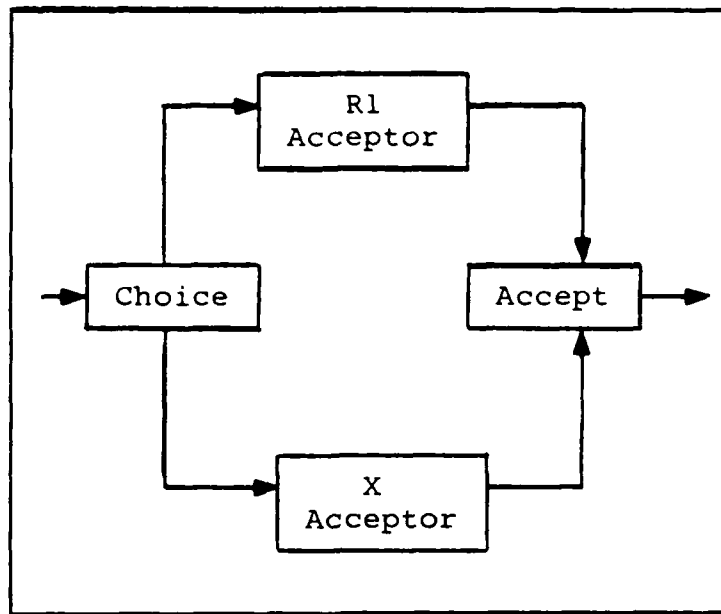


Figure 15.  $Y^+$  Acceptor

alphabet, the states and interconnecting paths, including the initial and final states used to accept the relation-words, can be identified within the automaton. This set of states and paths can then be investigated separately from the set accepting the node-label words. Using construction techniques, nondeterministic finite automaton is developed that will accept the relation-words (see Figure 15).

First, select a single relation  $R1$  from the relation-words  $Y^+$ . Next, construct a set  $X$  such that  $X = Y^+ - \{R1\}$ . Because  $Y^+$  and  $\{R1\}$  are regular sets over  $Y$ , the difference closure property of regular sets yields that  $X$  is also a regular set (92:186).

Now, let X-Acceptor and R1-Acceptor be the names of the finite automata that accept X and {R1}, respectively. Next, apply the union closure property of regular sets (92:186) and the equivalence in machine "power" of a nondeterministic finite automaton and a deterministic finite automaton (93:62) and construct an equivalent nondeterministic finite automaton accepting the relation-words  $Y^+$ . From Figure 15, this construction is accomplished by connecting the R1-Acceptor and X-Acceptor in a parallel fashion via a common initial state (the "choice" state of nondeterministic machines). Therefore, as words from  $Y^+$  are entered, the equivalent automaton effectively executes both the R1-Acceptor and X-Acceptor in parallel during the acceptance process. The overall automaton halts in the final "accept" state when either the R1-Acceptor or X-Acceptor accepts the input relation-word.

Next, since SN is assumed to be a type-3 language and accepted by some finite automaton, use this equivalent nondeterministic finite automaton as the mechanism that accepts the  $Y^+$  words during SN acceptance (see Figure 16). That is, replace the states and inter-

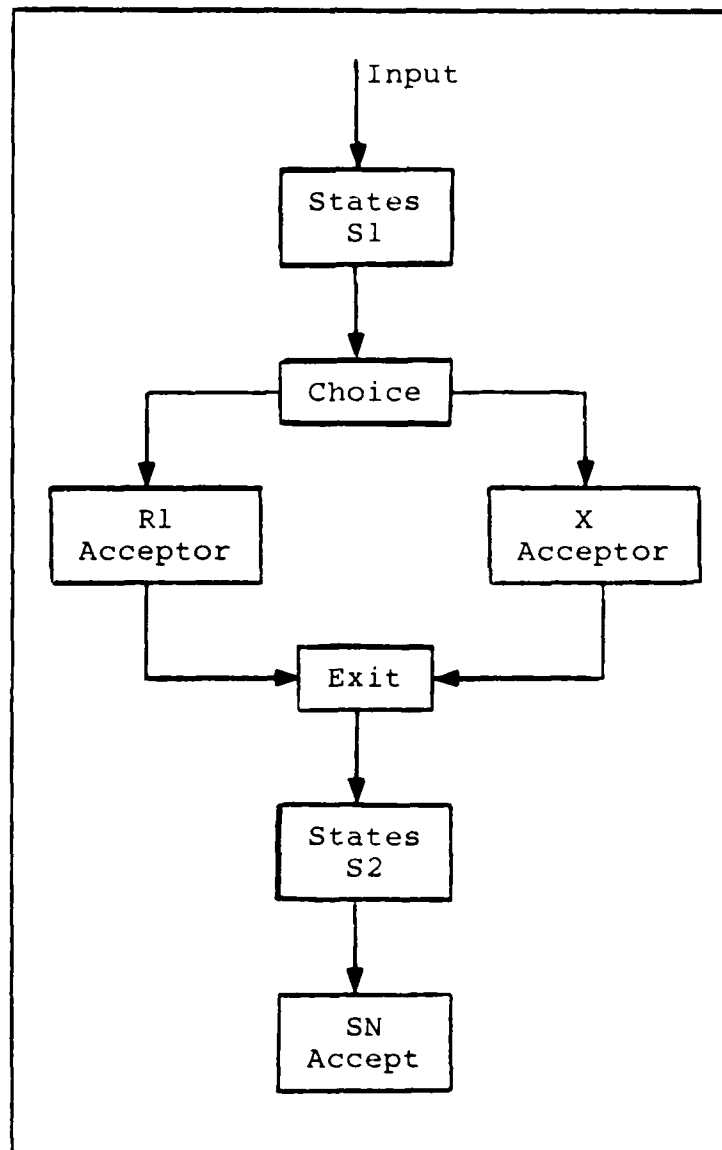


Figure 16. SN Acceptor

connecting paths previously used to accept  $y^+$  in the SN acceptor with this equivalent nondeterministic finite automaton. Because the SN machine does not necessarily halt upon accepting the relation-word, change the accept state to an "exit" state from the R1-Acceptor and the X-Acceptor. Since neither changing



the number finite states nor adding nondeterminism to a finite automaton changes the overall power of the machine (93), this exchange of  $Y^+$  acceptors does not change the power of the original SN acceptor automaton.

In this exchange of  $Y^+$  acceptors, the initial/final states of the  $Y^+$  acceptor are the final/initial states for the  $S_1$  and  $S_2$  sets of states, respectively. This has to occur because the incoming and outgoing paths to/from the  $Y^+$  acceptor were not modified.

After this new  $Y^+$  acceptor is in place in the SN acceptor, words from SN are now constrained to follow one of the two paths when being accepted. That is, in operation, the overall SN acceptor can really be considered as passing through two independent sequences of states depending on the relation-word in the overall semantic-network word. For example, words of the form  $aR_1b$  contained in SN can only be accepted by traversing the states of the  $R_1$ -Acceptor. Recognizing this true separation of paths, the proof can now be finalized.

Because of this guaranteed separation of paths, this overall SN acceptor (Figure 16) can be replaced by an equivalent acceptor.

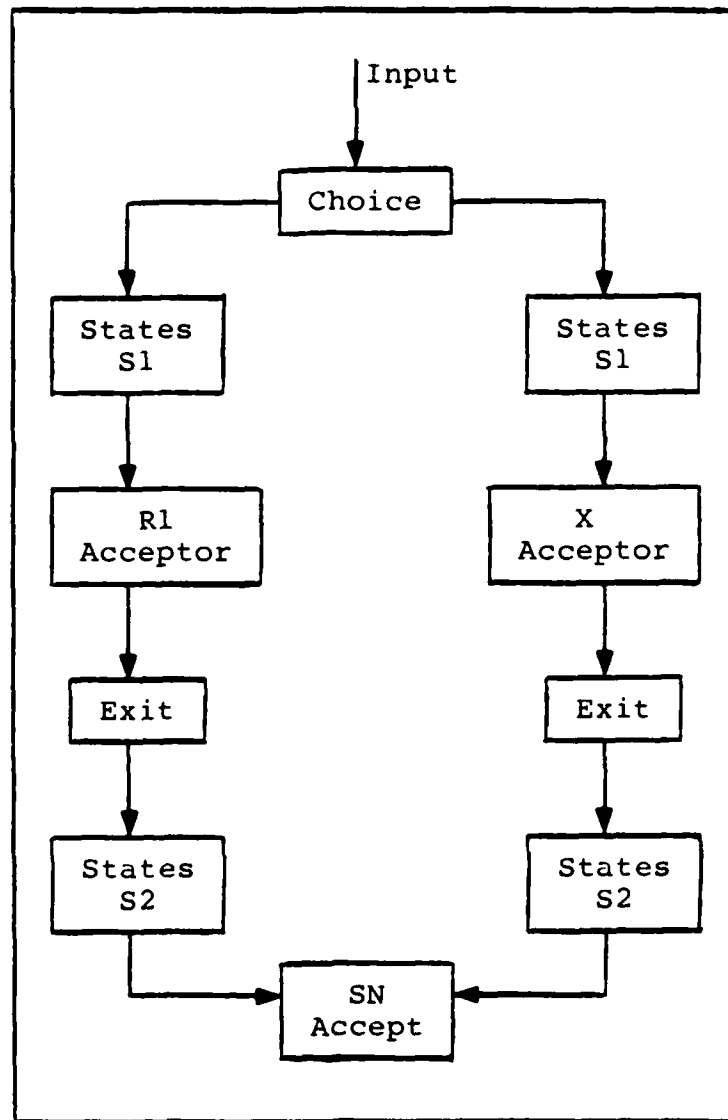


Figure 17. Equivalent SN Acceptor

This equivalent acceptor contains an independent path for words of the form  $aRb$  and an independent path for the words of the form " $aW(i)b$ " where  $W(i) \neq X$  (see Figure 17). This acceptor is constructed as follows: The choice state is moved from the relation-word subautomaton and placed as the initial state

in the overall machine. The original machine states (finite in number) are then duplicated in each of the choice paths, except that the R1-Acceptor is used in one path and the X-Acceptor is used in the other. Because the initial choice state was moved, the initial states in the R1-Acceptor and X-Acceptor become the new final states for the S1 set of states. This does not affect the operation of the acceptors nor the set of S1 states since the choice state effectively passed the same set of incoming paths to the initial states inside the two acceptors anyway. Also, the exit state is duplicated for both acceptors. The exit state remains both the final state for the acceptors and the initial state for the S2 states as before.

Therefore, this equivalent SN acceptor is a nondeterministic finite automaton. The automaton accepts words of the form  $aRlb$  by following one set of independent states and accepts words of the form  $aW(i)b$  by following another set of independent states. Even though there are duplicate states in each of the two paths, these states are not interconnected in any manner. Because of the placement of the choice state in the SN

acceptor, the two paths are effectively each a finite automaton accepting just words of the form  $aRlb$  and  $aW(i)b$ , respectively. In other words, since  $SN$  is assumed to be a regular set, the difference and union closure properties of regular sets can be used to form two other regular sets,  $\{aRlb\}$  and  $\{aW(i)b\}$ , whose union yields  $SN$ . Hence, the two separate automata accepting words of the form  $aRlb$  and  $aW(i)b$ , respectively, can be connected in parallel to accept the overall  $SN$  set.

Now, let  $m=Rl$  and let the arbitrary fact-alphabet and the arbitrary node-label alphabet be identical. If  $m$  is allowed semantically to be the implies relation of  $PR$ , this mapping results in this  $\{aRlb\}$  set becoming the  $PR$  language. So, this  $SN$  acceptor accepts the  $PR$  language as a special subset. But, this subset is actually accepted by a finite set of independent states in  $SN$ : a finite subautomaton. This  $PR$  set was previously proven not to be a type-3 (regular set) language. Hence,  $PR$  cannot be accepted by any finite automaton (see Theorem 5.05 of Section V). This is a contradiction of the regular set closure properties used to construct the  $\{aRlb\}$  regular set. Therefore, this finite

automaton (Figure 17) cannot accept all of the words in SN as previously assumed.

While this contradiction was found by constructing only one finite automaton, the contradiction condition also applies to any finite automaton that could be constructed to accept SN. The subautomaton that was constructed to accept  $Y^+$  (see Figure 15) was based on the known closure relationships of the regular sets and the known equivalence of nondeterministic/deterministic finite automata. Hence, it follows that any other finite subautomaton constructed to accept the relation-words  $Y^+$  has to be equivalent to the one constructed for this proof. That is, no matter how the  $Y^+$  regular set is partitioned into regular subsets (subsets must remain regular to be accepted by a finite automaton), there always exists one subset where the difference closure property can be applied to construct the single symbol,  $\{R1\}$ , regular set. Then, using the union closure property and nondeterminism, the equivalent separate path for  $\{R1\}$  acceptance can be shown to exist and the contradiction condition can be identified.

In addition to this equivalence of all possible  $Y^+$  acceptors, the accepting mechanism

for the node-label words  $Z^+$  and the  $a \neq b$  constraint have been adequately handled by the abstract nature of the construction. That is, the node-label words  $Z^+$  were not manipulated in any way by this construction. The accepting mechanism for them was handled abstractly in that two abstract sets of finite states,  $S_1$  and  $S_2$ , were assumed to perform the recognition of the node-label words as well as the  $a \neq b$  criterion. Therefore, all possible finite automaton acceptors have been considered and the recognition of the node-label words as well as the  $a \neq b$  criterion.

Therefore, all possible finite automaton acceptors have been considered and so SN has been proven not to be a type-3 language.

Q.E.D.

The proof that SN is not a type-3 language relied on SN's relationship to Haines' centermarker-languages (98). By applying this relationship again, the actual language-type of SN can be determined.

Theorem 6.06: The semantic network consisting of all the words that meet Definition 4.02 of Section IV is a type-2 language.

Proof: The proof of this theorem is by construction. The construction relies heavily on Haines' centermarker-languages and the type of machine

that accepts them (98). First, since SN is over  $Z^+$ , let M be the nondeterministic, single pushdown-store, machine that accepts the centermarker-language  $CM = \{xcy \mid x, y \in Z^+, x \neq y, c \notin Z\}$ . From the work of Haines, this machine must exist.

Next, make use of the fact that c is a disjoint, regular set and identify the finite set of states and interconnecting paths in M that are used to accept c. Since this centermarker is only a single symbol, it may be the case that a single path between two apparently unrelated states is used for this acceptance. For example, Haines used the c symbol to change between the x and y acceptance schemes. The symbol was effectively accepted by traversing a path between the states used in the acceptance of other unrelated words.

However, these apparently unrelated states are actually the initial and final states of the subautomaton that accepts c. The key point is that the initial and final states of the automaton accepting c may actually perform dual roles inside M. It is important to include these dual-function states in the subautomaton accepting c.

Putting aside for the moment this mechanism that accepts  $c$ , the characteristics of the relation-set that are identical to the characteristics of the  $c$  centermarker are determined. The relation-set is a countably infinite, disjoint, regular set that does not contain the empty word (see Theorem 6.02). From Definition 3.21 of Section III, this relation-set must be accepted by some finite automaton. Hence, both the relation-set and the  $c$  centermarker are accepted by finite automata.

Another characteristic that the relation-set and the  $c$  centermarker share is that neither one contains the empty word. Hence, a relation-word must always exist in the words of SN just as  $c$  must always exist in the words of CM.

The third and final identical characteristic between the relation-set and  $c$  centermarker is that the relation-set is constructed from an alphabet that is disjoint from the node-label alphabet. Therefore, the relation-words are always easily identified from the node-label words. That is, the accepting states and paths of the relation-set are independent of the accepting states and paths



for the node-label words. The  $c$  centermarker also has this characteristic.

Based on these common characteristics and the proof of Theorem 6.05, it follows that by replacing the finite set of states and interconnecting paths accepting  $c$  with the finite set of states and paths accepting the relation-set,  $M$  then accepts words that contain relation-words as the centermarkers. Therefore, let  $M'$  be the machine  $M$  modified by replacing the previously identified states and paths used to accept  $c$  with a finite automaton that accepts  $Y^+$ .

It is important to notice that this replacement action did not modify either the incoming paths to the initial state nor the outgoing paths from the final states of the  $c$  acceptor. This means that the previously mentioned dual functions that some initial/final states of the  $c$  acceptor may perform remain unchanged. Additionally, the replacement action did not change any other of the states and interconnecting paths involving the  $Z^+$  words. It follows then that the  $x \neq y$  constraint on the original set of words remains unchanged. Hence,  $M'$  actually accepts the words of  $SN$ .

Finally, as long as the number of states remains finite, the power of any type of machine is not affected by changing the number of states (93). Therefore,  $M'$  still remains in the class of nondeterministic, single pushdown-store, machines. From reference 8, page 43, it follows that SN is at least a type-2 language. Applying Theorem 6.05, SN is found to be exactly a type-2 language. Q.E.D.

So, SN has been proven to be a type-2 language. By considering both the node-label and relation sets as languages themselves, this semantic-network language has been shown to be dependent on both the network's structure and the knowledge-language (see Definition 4.04 of Section IV) it contains. More is discussed about the characteristics of SN later. But first, consider the case where the node labels and the relations are defined to be symbols in an alphabet.

Case II: Symbol Approach. In order to "hide" the language characteristics of the node labels and relations in the semantic-network strings, the node labels and relations are treated as arbitrary entities. To achieve this, each node label and relation is considered to be a single alphabet-symbol of unit length. However, the disjoint nature of the node-label and relation sets is still maintained. Also, because only the individual characteristics of the node-label and relation words are hidden, each

alphabet of node labels and relations remains countably infinite in size. Otherwise, the overall semantic-network set does not contain all the possible node-label words in  $Z^+$  and all the possible relation-words in  $Y^+$ . Using these changes, the definition of a semantic-network is modified as follows:

Definition 6.01: For any two arbitrary, disjoint, countably infinite sets, LB and R, a semantic network that has only category-1 characteristics (see Tables IV, V and VI of Section IV) is defined as a collection of 3-tuples. Each 3-tuple is of the form "arb" where "a" and "b" represent individual node labels, "r" represents the relation between the nodes,  $a, b \in LB$ ,  $r \in R$ ,  $|a|=|b|=|r|=1$  and  $a \neq b$ . The form arb reads "a is related to b by r." That is, the link arrow goes from a to b.

With this definition in hand, the first step is to determine if arb is a formal word.

Theorem 6.07: Every semantic network constructed under the constraints of Definition 6.01 consists of only strings of the form arb that are not finite words.

Proof: This follows directly from the definition of a word. While the length of each string is three and hence finite, it follows from Theorem 3.05 of Section III that the

overall alphabet,  $LB \cup R$ , is countably infinite. From Definitions 3.01 and 3.02 of Section III, words are constructed from only finite alphabets. Therefore, the strings are not finite words. Q.E.D.

Similar to the production-rule development, the arbitrary node-label and relation conditions applied to the semantic network force the strings not to be formal words. Since a language has to have words, this countably infinite alphabet leads to the following language theorem.

Theorem 6.08: The semantic network, SA, consisting of all possible strings meeting the Definition 6.01, is not a formal language.

Proof: Because languages require finite words as members, SA can be proven not to be a formal language immediately by applying Theorem 6.07. However, to emphasize the critical nature of the size of the alphabet, an alternate proof is given.

Recall from Definition 3.07 of Section III that a language is the set of all words that can be generated from a grammar. From the definition of a grammar (Definition 1.02 of Section I), a grammar has to have a finite set  $T$  of terminal symbols. The overall alphabet,  $LB \cup R$ , is this terminal-set  $T$  for the SA grammar. But, this alphabet is countably

infinite. Therefore, a grammar does not exist that generates all the finite strings (words) that are members of SA; thus SA cannot be a type-0 formal language. Q.E.D.

As was the case for production rules, this theorem also emphasizes that the finite length constraint on the symbol strings is only a necessary but not sufficient condition to make the strings formal words. This necessary condition is emphasized not only for the case where both of the sets LB and R are finite, but is also emphasized for the case where either the node labels or relations, but not both, are considered to be words in a language.

Theorem 6.09: Let either the node-label set or the relation-set consist of all the words (except e) over a nonempty, finite alphabet V and let the remaining set be comprised of a countably infinite set of symbols LB or R as needed. Then, the semantic network, SK, formed from all the possible strings meeting Definition 6.01, except  $a, b \in V^+$  (or LB) and  $r \in R$  (or  $V^+$ ), does not contain finite words.

Proof: This follows directly from the definition of a word. While the length of a string is three and hence finite, the overall alphabet, either  $LB \cup V$  or  $V \cup R$ , that the strings are constructed from is countably infinite. From Definitions 3.01 and 3.02 of Section III,

words are constructed from only finite alphabets. Therefore, the strings are not finite words. Q.E.D.

Hence, the SK semantic network cannot contain finite words. This lack of formal words leads to the next theorem regarding the language-type of SK.

Theorem 6.10: Let either the node-label set or the relation-set consist of all the words (except  $e$ ) over a nonempty, finite alphabet  $V$  and let the remaining set be comprised of a countably infinite set of symbols  $LB$  or  $R$  as needed. Then, the semantic network,  $SK$ , formed from all the possible strings meeting Definition 6.01, except  $a, b \in V^+$  (or  $LB$ ) and  $r \in R$  (or  $V^+$ ), is not a formal language.

Proof: Recall from Definition 3.07 of Section III that a language is the set of all words that can be generated from a grammar. From the definition of a grammar (Definition 1.02 of Section I), a grammar has to have a finite set  $T$  of terminal symbols. The overall alphabet,  $LB \cup V$  or  $V \cup R$ , is this terminal-set for the  $SK$  grammar. But, this alphabet is countably infinite. Therefore, a grammar does not exist that generates all the finite strings (words) that are members of  $SK$ ; thus  $SK$  cannot be a type-0 formal language. Q.E.D.

Theorems 6.08 and 6.10 reveal that when at least one of the disjoint sets of node labels and relations is constructed from an arbitrary, countably infinite alphabet, the semantic network cannot be a formal language. This is but one of the characteristics that have to be taken into account when attempting to map knowledge representations into formal languages. There are other characteristics that affect the SN, SA and SK semantic networks as well.

#### Semantic-Network Characteristics

Because of the abstract nature of the SA semantic-network definition (see Definition 6.01 and Theorem 6.08), the characteristics of SA are covered first. Since the SK semantic network (see Theorem 6.09) is only a variation on the SA network, it is not considered any further in this evaluation. Should the characteristics of SK be needed, they can be found by modifying the SA findings with the either/or case approach as was done in Theorems 6.09 and 6.10. However, the characteristics of the SN semantic-network language (see Theorem 6.06) are presented herein.

SA Characteristics. To investigate the SA semantic network further, the size of SA is determined.

Theorem 6.11: The SA semantic network, consisting of all possible 3-tuples of the form arb meeting Definition 6.01, is countably infinite.

Proof: Because the alphabets LB and R are each countably infinite, the proof of Theorem 6.02

can be applied directly to this theorem. That is, let the  $C_0, C_1, \dots$  of Figure 13 be members of LB instead of  $Z^+$ . Then, let  $R_1$  range over R instead of  $Y^+$ . Therefore, it follows that SA is countably infinite. Also the subset of words where  $a=b$  is countably infinite for the SA case as well. Q.E.D.

For reasons similar to those given for the PA production-rule set, Theorem 3.17 of Section III cannot be used to prove Theorem 6.11. However, the strings contained in SA can be forced to be words by restricting both LB and R to be finite sets.

Theorem 6.12: For an arbitrary, nonempty, finite subset of LB, LBF, and an arbitrary, nonempty, finite subset of R, RF, all of the strings that meet Definition 6.01, except that the strings are constructed from LBF and RF, are finite words.

Proof: Because LBF and RF are nonempty and finite, the overall alphabet-set of the semantic-network strings,  $LBF \cup RF$ , is nonempty and finite. Therefore, these semantic-network strings meet the first of the two constraints for formal words (see Definition 3.01 of Section III).

The second constraint, finite length (see Definition 3.02 of Section III), is met by the constraint that  $|a|=|b|=|r|=1$ . Hence, each



string is of length three, a finite number.  
Therefore, each string is a finite word over  
 $LBF \cup RF$ . Q.E.D.

Since the semantic-network words now exist, the search for a language-type can begin. But, this search is very short in comparison to the previous search for the SN language-type.

Theorem 6.13: For an arbitrary, nonempty, finite subset of LB, LBF, and an arbitrary, nonempty, finite subset of R, RF, the semantic network, SF, consisting of all the semantic-network words meeting Definition 6.01, except that the words are over LBF and RF, is a finite set and hence a type-3 language.

Proof: Let the size of LBF be a nonzero, finite number, J. Also let the size of RF be a nonzero, finite number, K. Following the same technique of matrix construction used in Theorem 5.02 of Section V and Theorem 6.02, construct the SF set as a three-dimensional matrix. That is, for each of the K relations, construct a two-dimensional matrix similar to Figure 14 as follows: For each relation, replace m in Figure 14 by the individual relation and let  $C_0, C_1, \dots, C_M$  represent the node labels in LBF. Let the M value in Figure

14 be equated to J. Finally, let the XX symbol still represent the  $a=b$  cases.

For this three-dimensional matrix, each of the two-dimensional matrices contain a total of  $J(J-1)$  entries. Since there are K two-dimensional matrices, the overall matrix consists of  $J(J-1)K$  entries. But, J and K are finite numbers. Therefore,  $J(J-1)K$  is a finite number and so SF is a finite set.

Using this finite-size characteristic, Theorem 6.12, and applying Definitions 3.19, 3.21 and Theorem 3.12 of Section III, SF is a found to be a type-3 language. Q.E.D.

By constraining the node-label and relation sets to be finite, a formal language can be obtained for the case where node labels and relations are symbols. However, Theorem 6.13 only deals with a single arbitrary case of converting a subset of strings in SA into some formal language. The next theorem expands this finding to cover a class of languages.

Theorem 6.14: There exists a countably infinite number of semantic-network languages where each language is finite in size and consists of all the words meeting Definition 6.01, except that the node labels and relations are selected from arbitrary, nonempty, disjoint, finite sets of symbols.

Proof: Theorem 6.13 was based on selecting a single, finite subset of node labels from LB and a single, finite subset of relations from R to generate a semantic-network language that is finite in size. Applying Theorem 3.10 of Section III, there are a countably infinite number of finite subsets that can be selected from both LB and R. Now, from set theory, the empty set  $\phi$  is a finite subset of any set. Hence,  $\phi$  is a member of each of these countably infinite sets of node labels and relations.

Because of the nonempty-constraint, these empty sets must be removed from the sets of node labels and relations. By applying the  $f(x)=x-1$  mapping on the indexes of the remaining finite sets of both node labels and relations, there still exists a countably infinite number of finite subsets of node labels and relations.

Now, construct the matrix given in Figure 18, where the rows represent the relation-set used to generate a given semantic-network language and the columns represent the associated node-label set. That is, each entry represents a node-label and relation set that

	LBF0	LBF1	LBF2	..
RF0	RF0,LBF0	RF0,LBF1	RF0,LBF2	..
RF1	RF1,LBF0	RF1,LBF1	RF1,LBF2	..
RF2	RF2,LBF0	RF2,LBF1	RF2,LBF2	..
..	..	..	..	..

Figure 18. Node-Label and Relation Sets

can be used to generate a finite semantic-network language.

Since there is a countably infinite number of finite node-label sets, each row in this matrix has a countably infinite number of entries. By performing the union operation on the countably infinite number of rows and applying Theorem 3.05 of Section III, there is a countably infinite number of unique entries in the matrix. Therefore, there exists a countably infinite number of unique semantic-network languages that are finite in size.

Q.E.D.

Because finite languages have been studied extensively by Maurer et al. (99; 100), further investigation of the characteristics of these finite semantic-network languages is not provided. However, it is important to notice that when the node labels and relations are selected from arbi-

trary, countably infinite, disjoint symbols, then no single semantic-network language exists that can contain all the possible semantic-network words. See Section IX for additional affects of the node-label and relation languages on the semantic-network languages.

Category 1 Class. Since the SN version of the semantic network does contain all possible semantic-network words for given Z and Y alphabets, the characteristics of the SN language merit further investigation. Due to the fact that the general characteristics of context-free languages have been studied extensively, the common characteristics of type-2 languages (e.g. normal grammar forms, word membership algorithm, parsing algorithm) that apply to SN are not investigated further. The reader is referred to references 6, 92, 93, 94, 95, 96 and to the references contained in Appendix A for information on how to determine these individual type-2 characteristics.

However, there are characteristics of a class of semantic-network languages that contains the SN language that directly affect later findings of this investigation. The characteristics of this class are covered in depth.

Theorem 6.15: Any semantic network consisting of all the words meeting Definition 4.02 of Section IV, except that the node-label set and the relation-set are any nonempty, regular sets over Z and Y, respectively, is at least a type-2 language.

Proof: This proof follows directly from Haines (98) and the proofs of Theorems 6.06 and 6.14. Haines proved that for any centermarker-language where the  $x$  and  $y$  strings of the centermarker-word  $xcy$  are selected from any regular set, the resulting language is type 2. By applying the constraint of a nonempty, regular set on the node labels, Haines' necessary condition for the type-2 language finding is met. The nonempty-constraint comes from the network's definition and does not affect Haines' theorem in any way.

However, the constraint that the centermarker is only a single symbol is not met. Theorem 6.06 resolved this constraint for one regular relation-set by showing that the same type of machine (not identical machines) that accepts Haines' centermarker-languages also accepts the SN language. The key in that proof was the replacement of the  $c$  accepting mechanism with the finite automaton that accepts the  $Y^+$  relation-words.

Since Definition 3.21 of Section III establishes that all regular sets are accepted by some finite automaton, the proof of Theorem 6.06 can be expanded to include any nonempty

regular set of relation-words. That is, replace the  $c$  accepting mechanism with the specific finite automaton that accepts the nonempty, regular relation-set. The nonempty-constraint maintains the condition that a centermarker must exist for these words. Since the power of a machine does not change by adding (subtracting) a finite set of states to (from) a machine (93), the type of machine does not change for any nonempty, regular relation-set. Hence, these semantic-network languages are upper bounded at type 2.

Now, to show the "at least" constraint, Theorems 6.06 and 6.14 are needed. Theorem 6.06 established that SN is exactly a type-2 language. Since SN uses nonempty, regular, node-label and relation sets, SN must be contained in the class of category-1 languages covered by this theorem.

Next, Theorem 6.14 established the existence of a countably infinite set of type-3 semantic-network languages where the node labels and relations are selected from disjoint, nonempty, finite regular sets. If  $Z^+$  and  $Y^+$  are used as the countably infinite sets of symbols in Theorem 6.14, then these finite languages are also contained in the class of

category-1 languages covered by this theorem. Therefore, since both type-2 and type-3 semantic-network languages are contained in this class of category-1 languages covered by this theorem, it follows from Chomsky's language-hierarchy (8) and the upper type-2 language bound that these semantic-network languages are at least type-2. Q.E.D.

As seen from Theorem 6.15, SN is only one of a number of possible semantic-network languages each of which has only category-1 characteristics (see Tables IV, V and VI in Section IV). The actual size of this class of languages provides another important characteristic.

Theorem 6.16: The SN semantic-network language, consisting of all the words meeting Definition 4.02 of Section IV, is a member of class of languages that consists of a countably infinite number of type-2 semantic-network languages.

Proof: This proof follows from Theorem 3.18 of Section III and Theorems 6.02, 6.06, 6.14 and 6.15. Theorem 6.15 established that type-2 semantic-network languages can be constructed provided the node labels and relations are selected from nonempty, disjoint, regular sets. Theorems 6.02 and 6.06 revealed the existence of at least one language that is



countably infinite in size, SN, that meets the constraints of Theorem 6.15. By using  $Z^+$  and  $Y^+$  as the arbitrary, countably infinite sets, Theorem 6.14 shows the existence of a countably infinite set of semantic-network languages each of which is finite in size and also meets the regular-set constraints of Theorem 6.15. Hence, since SN is countably infinite in size, SN cannot be contained in this countably infinite set of finite languages. From set theory, the union of a finite set and a countably infinite set is countably infinite. So, SN is a member of a set of languages that is at least countably infinite in size.

To prove the upper bound, the size of language space is needed. From Theorem 3.18 of Section III, there exists a countably infinite number of languages over a given alphabet. Therefore, the upper bound for this class of category-1 languages is countably infinite. Since both the upper and lower bounds agree, this class of semantic-network languages is countably infinite in size.

Q.E.D.

So, at the category-1 level of the definition-hierarchy (see Figure 12 in Section IV) there exist countably infinite

numbers of semantic-network languages of which SN is a member. The fact that this class consists of all possible languages over the regular node-label and relation sets yields a further characteristic about this class of category-1 languages.

Theorem 6.17: The class of category-1 languages consisting of semantic-network languages where each language consists of all the words meeting Definition 4.02 of Section IV, except that the node-label and relation sets for each language are any nonempty, regular sets over  $Z$  and  $Y$ , respectively, is not closed under the union operation.

Proof: This proof follows from construction of a counter-example. For this proof, arbitrarily select the  $Z$  and  $Y$  alphabets as follows:  $Z=\{c, d, g, h, k\}$  and  $Y=\{p, s, t, x, y\}$ . For the  $Z$  alphabet, arbitrarily select two finite, regular sets as follows:  $S1=\{c, d\}$  and  $S2=\{g, h\}$ . For the  $Y$  alphabet, arbitrarily select two finite, regular sets as follows:  $T1=\{p, t\}$  and  $T2=\{x, y\}$ . Now, construct two semantic-network languages using  $S1-T1$  and  $S2-T2$  as the node-label and relation sets:  $L1=\{cpd, dpc, ctd, dtc\}$  and  $L2=\{gxh, hxg, gyh, hyg\}$ . Finally, using these two languages, form a third language by

the union operation:  $L3 = \{cpd, dpc, ctd, dtc, gxh, hxg, gyh, hyg\}$ .

By this construction,  $L1$  and  $L2$  meet the constraints needed to be members of this class of semantic-network languages. That is, each of these two languages consists of all the words meeting Definition 4.02 of Section IV except that the node labels and relations are selected from some nonempty, regular sets over  $Z$  and  $Y$ , respectively. However, the language formed from their union,  $L3$ , does not meet these constraints.

The  $L3$  language contains semantic-network words formed from regular sets of four individual node labels,  $S3 = \{c, d, g, h\}$  and four individual relations,  $T3 = \{p, t, x, y\}$ . But,  $L3$  does not contain all possible semantic-network words over the  $S3$  node-label set and the  $T3$  relation-set. For example, the legal semantic-network word "cxg" is not contained in  $L3$ . Yet, by definition of the semantic-network languages contained in this class of category-1 languages, this specific word (and others) must be contained in a semantic-network language over  $S3$  and  $T3$ . Therefore,  $L3$  is a finite language but is not a semantic-network language over these regular sets.

Since the union closure property requires that  $L_3$  be in the class, it follows that this class of category-1 languages is not closed under the union operation. Q.E.D.

As was the case for the class of category-1 production-rule languages (see Theorem 5.16 of Section V), this union closure property is not the only regular operation that this class of semantic-network languages does not meet. The class also does not meet the product closure property. (The reader is referred to the paragraph preceding Theorem 5.16 of Section V for the definition of product closure).

Theorem 6.18: The class of category-1 languages consisting of semantic-network languages where each language consists of all the words meeting Definition 4.02 of Section IV, except that the node-label and relation sets for each language are any nonempty regular sets over  $Z$  and  $Y$ , respectively, is not closed under the product operation.

**Proof:** This proof follows from the construction of a counter-example. For any two semantic-network languages in this class of category-1 languages, this product operation will generate strings of the form "A1R1B1A2R2B2." Because a semantic-network word has only one relation-word, the product operation generates illegal network-words. Hence, the strings generated

by the product operation cannot be contained in any semantic-network language. So, this class of category-1 semantic-network languages is not closed under product. Q.E.D.

These nonclosure properties of this class of semantic-network languages lead to the last characteristic of this class of languages that is presented in this section. This characteristic is that this class of semantic-network languages does not form an Abstract Family of Languages.

Theorem 6.19: The class of category-1 languages consisting of semantic-network languages where each language consists of all of the words meeting Definition 4.02 of Section IV, except that the node-label and relation sets for each language are any nonempty regular sets over  $Z$  and  $Y$ , respectively, is not an AFL.

Proof: From reference 24, an AFL contains languages that are closed under six operations: union, product, Kleene closure, e-free homomorphism, inverse homomorphism and intersection with a regular set. From Theorems 6.17 and 6.18, this class of category-1 semantic-network languages is not closed under either union or product. Therefore, this class of category-1 semantic-network languages cannot be an AFL. Q.E.D.

The result of Theorem 6.19 presents a major stumbling block to the further study of this class of category-1 semantic-network languages. Because this class is not an AFL, the determination of additional characteristics of this class has to involve the analysis of the entire set of network-languages in the class. Thus, the multitude of the characteristics proven about AFLs by formal-language theorists cannot be used to help expand the characteristic set of this class of languages.

The characteristics of AFLs are not the only characteristics that do not apply to this class of languages. The characteristics of the general class of type-2 languages also do not apply. For example, the entire set of Chomsky's type-2 languages forms an AFL but this class of category-1 semantic-network languages does not. Another example is that the entire set Chomsky's type-2 languages is closed under the regular operations of union and product (6:12), yet this class of languages is not.

So, while this class of category-1 semantic-network languages is a set of type-2 languages, the characteristics of the general type-2 languages cannot be used, without proof, when referring to this class of languages. Even though other characteristics of this class of languages as well as the characteristics of other classes of category-1 semantic-network languages could be investigated, sufficient information is now known such that the comparison of the production-rule and semantic-network languages can begin.

## VII. Comparison of Rules and Networks

In Sections V and VI, the characteristics of the production-rule and semantic-network languages were found to be similar. At the category-1 level of the definition hierarchies (see Section IV), there exist special classes consisting of countably infinite numbers of production-rule and semantic-network languages. Each language in those classes is at most a type-2 language. However, each of the classes itself is not an Abstract Family of Languages (24). Because of these similarities, the question of equivalence (one-for-one and onto) of these representation languages comes to the forefront of this investigation. Therefore, this section begins by investigating the production-rule and semantic-network language equivalence, then, transformations between these languages are analyzed.

### Equivalences

The question of equivalence between these two representation languages can be answered for two cases. The first case is the trivial one where the implies relation does not exist as a semantic-network relation and the bijective mapping  $R1=m/m=R1$  between one word in the relation-set and the symbol  $m$  is not allowed. Because the fact-set and the node-label set are arbitrary, these two sets can be mapped into one another by using alphabets of the same size.

However, if the  $m$  and relation centermarkers have no common mapping, then these two languages cannot be equivalent.

The second case is where either the implies relation does exist as a semantic-network relation or the bijective mapping  $R_1 = m/m = R_1$  between one word in the set of relations and the symbol  $m$  is allowed. Under either one of these conditions, equivalence of special sets of languages are shown to exist. For clarity in stating theorems that depend on these mapping conditions, the conditions are restated in the following definition of "condition-A." The theorem statements will then refer to these mapping conditions by stating that the languages meet condition-A.

Definition 7.01: Production-rule and semantic-network

languages are said to meet condition-A if either the implies relation does exist as a semantic-network relation or the bijective mapping  $R_1 = m/m = R_1$  between one word in the relation-set and the symbol  $m$  is allowed.

Using this definition, the first of several equivalence theorems is given as follows:

Theorem 7.01: For the special classes of category-1

production-rule and semantic-network languages (see Theorem 5.13 of Section V and Theorem 6.15 of Section VI for class definitions), there exists a countably infinite set of equivalent pairs of finite production-rule and



finite semantic-network languages. The languages in this set meet condition-A.

Proof: This proof relies on construction. First, consider the fact and node-label sets. Because the production-rule's set of facts and the semantic-network's set of node labels are constructed from arbitrary, nonempty, finite alphabets, a simple bijective mapping between the alphabets can be used as the basis for mapping between these sets. This mapping consists of making the size of the alphabet symbols of these two sets the same and equating the alphabet symbols one-to-one. Then, by applying the regular set constraint on these isomorphic alphabets, isomorphic fact and node-label word-sets results.

Next, for each of these isomorphic alphabets, perform the e-free Kleene closure (i.e.  $Z^+$ ) and select all possible finite subsets. From Theorem 3.10 of Section III and the fact that these alphabets are isomorphic, this action results in the formation of a countably infinite number of pairs of isomorphic finite subsets. Because the empty set  $\emptyset$  is a finite subset of every set, one pair of these isomorphic sets involves the empty set. Since the fact and node-label sets cannot be

empty, this isomorphic pair must be removed from this group of isomorphic pairs. By applying the  $f(x)=x-1$  mapping to the indexes of the remaining pairs of isomorphic subsets, there is still a countably infinite number of pairs of isomorphic finite subsets.

Now that the finite fact and node-label sets have been isomorphically mapped, construct the finite set of relation-words needed for the centermarker mapping as follows. For the relation-alphabet  $Y$ , perform the  $\epsilon$ -free Kleene closure  $Y^+$ . From this countably infinite set, apply Theorem 3.10 of Section III to obtain the countably infinite number of finite subsets of relation-words. As before, remove the empty set  $\emptyset$  and apply the  $f(x)=x-1$  mapping to the indexes of the remaining subsets to obtain a countably infinite number of nonempty, finite relation-sets. Then, from these finite relation-sets, select the single-element set that either contains the implies relation or contains the relation where the bijective mapping of  $R_1=m/m=R_1$  is allowed (only one subset exists). This relation-set becomes the isomorphic centermarker for the two language versions.

With the fact, node-label and relation sets established, consider first the mapping of semantic networks to production rules. Construct the semantic-network languages as follows: Pair up each of the sets of node-labels previously constructed with the relation-set consisting of the isomorphic centermarker. Then, using each of these pairs, construct a finite semantic-network language (see Theorem 6.13 of Section VI for proof of the finite cardinality). The language is constructed to consist of all the words meeting the constraints of Definition 4.02 of Section IV, except the node-label and relation sets are finite sets of words. Because the node-label and relations sets are finite, each set is also a regular set (see Definition 3.19 of Section III). Therefore, each of these finite semantic-network languages is a member of the special class of category-1 semantic-network languages (see Theorem 6.15 for class definition). Also, since there is a countably infinite number of these finite node-label sets, there is a countably infinite number of finite semantic-network languages that are constructed.

Now, to each of these finite semantic-network languages, apply the bijective mapping  $m=R_1$  and exchange each node-label word with its previously constructed isomorphic fact-word. For each network-language, this action results in the generation of an equivalent, finite, production-rule language. Because each of the equivalent production-rule languages has an associated finite, and hence regular, set of facts, each rule-language is a member of the special class of category-1 production-rule languages. Since there is a countably infinite number of these finite semantic-network languages, there is a countably infinite number of production-rule languages generated.

Next, consider the mapping of production rules to semantic networks. Use each of the finite fact-sets constructed previously and construct the countably infinite set of finite production-rule languages (see Theorem 5.11 of Section V for proof of the finite cardinality). Each language is constructed to consist of all the words meeting the constraints of Definition 4.01 of Section IV, except the fact-set is a finite set of words. Since each set of facts is finite in size and hence

regular, each language is a member of the special class of category-1 production-rule languages.

Now, to each of these production-rule languages, apply the bijective mapping  $Rl=m$  and exchange each fact-word with its previously constructed isomorphic node-label word. For each rule-language, this action results in the generation of an equivalent, finite, semantic-network language. Because each of the equivalent semantic-network languages has finite, and hence regular, sets of node labels and relations, each network-language is a member of the special class of category-1 semantic-network languages. Since there is a countably infinite number of these finite production-rule languages, there is a countably infinite number of semantic-network languages generated.

Finally, because the rule-to-network mapping uses exactly the same isomorphic fact, node-label and relation sets as the network-to-rule mapping, the network (rule) language that is produced as a result of the rule-to-network (network-to-rule) mapping is identical to the network (rule) language used to generate the original rule (network)

language. Therefore, the rule-to-network and network-to-rule mappings produce pairs of equivalent, finite, production-rule and semantic-network languages. Hence, there exists a countably infinite number of pairs of equivalent, finite production-rule and finite semantic-network languages. Q.E.D.

Even though there is a countably infinite number of pairs of equivalent finite languages, there still exists a countably infinite number of finite semantic-network languages that are not equivalent to any of the finite production-rule languages.

Theorem 7.02: For the special classes of category-1 production-rule and semantic-network languages (see Theorem 5.13 of Section V and Theorem 6.15 of Section VI for class definitions), there exists a countably infinite number of finite semantic-network languages that are not equivalent to any finite production-rule language.

Proof: This follows directly from the proof of Theorem 7.01. In that proof, the isomorphic mapping between pairs of production-rule and semantic-network languages uses only one of the countably infinite, finite relation-sets for the centermarker mapping. For a given alphabet  $Z$  and  $Y$ , there still exist the other

semantic-network languages that can be constructed out of the remaining finite relation-sets. These languages are not equivalent to any finite production-rule language. This is shown in Figure 18.

In Figure 18, let the  $RF\emptyset$  relation-set be the single-element relation-set of Theorem 7.01. Next, let each  $LBF(i)$  represent one of the finite node-label sets of Theorem 7.01. Then, the row marked  $RF\emptyset$  represents the isomorphic set of semantic-network languages of Theorem 7.01. The remaining rows represent the other finite semantic-network languages that are not equivalent to any finite production-rule language. They cannot be equivalent because the relation-sets used to construct them either contain more than one relation-word or do not contain the necessary centermarker relation-word.

Continuing to refer to Figure 18, remove the  $RF\emptyset$  row from the matrix. Using the additional mapping of  $f(x)=x-1$  on the indexes of these remaining finite relation-sets, there still exists a countably infinite number of rows in this matrix. Since each row represents the pairing of a finite relation-set with each of the countably infinite number of

finite, node-label sets of Theorem 7.01, there is a countably infinite number of semantic-network languages possible per row. Applying Theorem 3.05 of Section III to the union of these remaining rows, there is a countably infinite number of finite, semantic-network languages that are not equivalent to any finite production-rule language. Q.E.D.

Now that the equivalence and nonequivalence cases for finite production-rule and semantic-network languages have been determined, the same cases for languages having countably infinite cardinality need investigating. First, the equivalence of the production-rule language, PR, and the semantic-network language, SN, is determined.

Theorem 7.03: The category-1 production-rule language, PR, that consists of all the words meeting Definition 4.01 of Section IV, is not equivalent to the semantic-network language, SN, that consists of all the words meeting Definition 4.02 of Section IV. These languages meet condition-A.

Proof: As was the case for the previous two theorems, this proof also relies on construction. First, let the fact and node-label alphabets used in PR and SN be bijectively mapped by equating their symbols as was done in Theorem 7.01. Then, the required closure,  $Z^+$ , yields



isomorphic fact and node-label word-sets between the rule and network languages.

Next, perform the  $Y^+$  closure on the relation-alphabet. In this set, identify the relation  $R_l$  where the bijective mapping to the  $m$  symbol is allowed. Using this  $R_l$  word, group together all the words in  $SN$  that contain  $R_l$  as their relation-word. Because of the isomorphic mapping of the fact and node-label sets and the  $R_l = m / m = R_l$  mapping, this group of  $SN$  words is equivalent to the  $PR$  language.

However, there exist additional words in  $SN$  that are not contained in this isomorphic group of words. This follows directly from the fact that  $R_l$  is a single word and  $Y^+$  contains countably infinite words. Therefore, there exist words in  $SN$  where the relation in the word is not bijectively mapped to  $m$ . Hence, all the words in  $SN$  cannot be bijectively mapped into all of the words of  $PR$  and so  $PR$  and  $SN$  cannot be equivalent. Q.E.D.

Even though  $PR$  and  $SN$  are not equivalent, there exists one category-1 semantic-network language that is equivalent to  $PR$ .

Theorem 7.04: There exists only one category-1 semantic-network language that is equivalent to the

production-rule language PR. These languages meet condition-A.

Proof: This proof is by construction. Construct a subset of the special category-1 semantic-network languages (see Theorem 6.15 of Section VI for class definition) as follows. For each semantic-network language in this subset, use the  $Z^+$  regular set as the node-label set and one of all the possible unique regular sets over Y for the relation-set.

Now, use the techniques of Theorem 7.01 to bijectively map the node-label alphabet to the fact-alphabet of PR. Then, the  $Z^+$  closure yields isomorphic fact and node-label word-sets. Because PR is constructed from only the  $Z^+$  regular set, only those network-languages in this specially constructed subset of network-languages can possibly be equivalent to PR.

Next, apply the bijective mapping of  $R1=m/m=R1$  to one and only one of the semantic-network's relations. Because each of the relation-sets is a unique regular set, there exists one and only one semantic-network language in this subset of network-languages that has the single R1 relation as its relation-set. Since this semantic-network

language is constructed over the required  $Z^+$  node-label set and the  $Rl=m/m=Rl$  mapping applies to the only relation-word in this network-language, it is the only category-1 semantic-network language that can be mapped to PR.

Finally, using the techniques of Theorem 7.01 and reversing the steps of this proof, PR can be shown to map to this identical semantic-network language. Hence, there exists only one category-1 semantic-network language that is equivalent to PR. Q.E.D.

By using different, countably infinite, regular subsets of  $Z^+$  for the node labels (e.g.  $Z^+ - \{F(i)\}$ ,  $F(i) \in Z^+$ ), Theorem 7.04 can be expanded to show that there exists a countably infinite set of pairs of equivalent production-rule and semantic-network languages. Each language in one of these pairs has countably infinite cardinality. However, by using different regular subsets of  $Y^+$ , it is shown that there exists a countably infinite set of semantic-network languages that are not equivalent to any production-rule language. The semantic-network languages in this set have countably infinite cardinality.

Theorem 7.05: There exists a countably infinite number of countably infinite category-1 semantic-network languages that are not equivalent to any

countably infinite category-1 production-rule language.

Proof: This proof is by construction. First, apply the bijective map between the fact-alphabet and the node-label alphabet and construct the isomorphic fact and node-label sets. However, there is an additional constraint on these sets. From the proofs of Theorem 5.02 of Section V and Theorem 6.02 of Section VII, any production-rule or semantic-network language that has countably infinite cardinality has a countably infinite fact-set or a countably infinite node-label (or relation) set, respectively. So, for the first part of this proof, select a countably infinite regular set for the isomorphic fact and node-label sets.

Next, using this countably infinite node-label set, construct a set of countably infinite semantic-network languages as follows. First, perform the  $Y^+$  closure on the relation-alphabet to generate all the relation-words. Then, for each relation  $R(j)$  in  $Y^+$ , construct the relation-set  $Y^+ - \{R(j)\}$ . Since  $Y^+$  and  $\{R(j)\}$  are each regular sets, applying the difference closure property of regular sets yields that each of these new relation-sets are also regular sets (92:186).

Because they are regular sets, they can be used to construct semantic-network languages that are contained in the special class of category-1 languages (see Theorem 6.15 for class definition). Each of these languages has countably infinite cardinality. Therefore, for each of these relation-sets and the given node-label set, construct a set of countably infinite category-1 semantic-network languages.

Now that a set of countably infinite size semantic-network languages has been constructed, the size of this set can be determined. Since  $Y^+$  has of a countably infinite number of individual relations  $R(j)$ , there is a countably infinite number of  $Y^+ - \{R(j)\}$  regular sets that can be constructed. Because each of these semantic-network languages is constructed from one of these regular relation-sets, it follows that this set of network-languages is countably infinite in size.

With the size of this set established, all that remains is to show that the semantic-network languages in this set cannot be equivalent to any production-rule language. First, by applying the  $f(x)=x-1$  mapping to the remaining relation-words in each of the

$Y^+ - \{R(j)\}$  relation-sets, each of these regular relation-sets is found to be countably infinite in size. Hence, no semantic-network language exists in this set of network-languages that has a relation-set of length of one.

Because of this characteristic, it follows from the proof of Theorem 7.03 that each semantic-network language in this set of network-languages contains words that cannot be isomorphically mapped to any words contained in any category-1 production-rule language. Therefore, there exists a countably infinite set of countably infinite category-1 semantic-network languages that are not equivalent to any category-1 production-rule language.

Finally, from Theorem 3.18 of Section III, because there only exists a countably infinite number of possible languages, there is at most a countably infinite number of unique node-label sets that can be used to construct other sets of these semantic-network languages. From Theorem 3.05 of Section III, the union of all these sets of semantic-network languages yields a countably infinite number of countably infinite category-1 semantic-network

languages that are not equivalent to any countably infinite category-1 production-rule language. Q.E.D.

With the proof of Theorem 7.05 complete, sufficient information now exists to reach the general conclusion on the equivalence of the special classes of category-1 rule and network languages.

Theorem 7.06: The special class of category-1 production-rule languages is not equivalent to the special class of semantic-network languages (see Theorem 5.13 of Section V and Theorem 6.15 of Section VI for class definitions).

Proof: Because finite languages cannot be equivalent to countably infinite languages, it follows from Theorems 7.02 and 7.05 that this special class of category-1 semantic-network languages contains finite and countably infinite languages that are not equivalent to any production-rule language. Therefore, the special class of category-1 semantic-network languages is not equivalent to the special class of category-1 production-rule languages. Q.E.D.

So, even though special cases of equivalent production-rule and semantic-network languages exist, these special classes of category-1 production-rule and category-1 semantic-network languages are not equivalent. Further

details of the impact of this finding are given in Sections VIII and IX. While these classes of languages are not equivalent, the question now arises about the ability to transform one type of language into another by any kind of mapping.

### Transformations

Transforming one language into another depends on the ability to convert the constrained words of one language into the constrained words of the other. Or, in equivalent fashion, the ability to transform the grammar generating one language into one that generates the new language. From the proofs of Theorems 7.01 and 7.04, the transformation from a production-rule language to a semantic-network language is seen to be straightforward. Therefore, the rule-to-network transformation is considered first.

Theorem 7.07: Any production-rule language contained in the special class of category-1 rule-languages can be transformed into an equivalent category-1 semantic-network language that is contained in the special class of category-1 network-languages (see Theorem 5.13 of Section V and Theorem 6.15 of Section VI for class definition) by the following transformation:

1. Equate the fact and node-label alphabets and word-sets.
2. Equate the single implies symbol  $\Rightarrow$  to a single relation-word that is over an



alphabet disjoint from the node-label alphabet and use this word as the entire relation-set.

3. Replace the terminal sets as described in steps 1 and 2 in the production-rule grammar and use the resulting grammar to obtain the semantic-network language.

Proof: Step 1. From Theorem 5.13 of Section V, any production-rule language in the special class of category-1 rule-languages has a regular set over  $Z$  for the facts. From Theorem 7.01, by assigning the fact-alphabet to be the same alphabet for the node labels, the regular set of facts becomes the node-label set. This causes the node-label set to become a regular set. Hence, the first requirement of a category-1 semantic-network language, a regular node-label set, is met.

Step 2. By assigning the  $m$  symbol to a single relation-word over an alphabet disjoint from the node-label alphabet and making this relation-word the entire relation-set for the network-language, another condition for the category-1 semantic-network language is satisfied.

This condition is that the relation-set be a regular set disjoint from the node-label alphabet.

Step 3. Word-for-word transformations can be applied to generate the category-1 semantic-network language with the constraint, "consists of all the words over the node-label and relation sets where  $a \neq b$ ," upheld. That is, by replacing the fact-words with the node-label words and the  $m$  symbol with the single relation-word, this constraint that is already existing in the production-rule language is automatically passed on to the semantic-network language. However, when the production-rule language has countably infinite cardinality, this word-for-word transformation never terminates. This causes, in turn, difficulty in proving that the transformation generates an equivalent semantic-network language. To overcome this problem, a grammar-to-grammar transformation is used.

Because the only difference between the category-1 production-rule and semantic-network words is the terminal set of symbols, a grammar-to-grammar transformation can be accomplished by simply changing the terminal

set in the production-rule grammar. That is, in each of the grammar's generative rules, replace  $m$  with the single relation-word  $R_1$  and use the identical fact and node-label alphabet for the remaining terminal symbols. Since the form of the grammar's generative rules is not changed by this grammar-to-grammar transformation, the  $a \neq b$  constraint on the production-rule words is passed on to the semantic-network word. Also, by maintaining the same form of grammar-rules, the modified grammar still generates exactly the same number of words as was contained in the original rule-language. Therefore, the set of words generated by this new grammar meet the last constraint, "consists of all the words over the node-label and relation sets where  $a \neq b$ ," of a category-1 semantic-network language.

Having proven that by following the steps of this transformation a semantic-network language can be obtained, all that remains is to point out specifically that the network-language is equivalent to the production-rule language. This follows from several conditions. One condition is that since the fact and node-label alphabets are identical, the regular word sets formed from these alphabets

are one-for-one and onto mappings. Another condition is that the  $R_l$  and  $m$  symbols are identity relations and therefore bijective. The final condition is that the modified grammar generates exactly the same number of words as was originally contained in the production-rule language. That is, finite languages transform one-to-one into finite languages and countably infinite languages transform one-to-one into countably infinite languages. This convergence is guaranteed because the original rule-language converged using the same form and number of generative rules.

From this proof of equivalence and the fact that each transformation begins with a given production-rule grammar (one is guaranteed to exist), it follows that every production-rule language contained in the special class of category-1 rule-languages can be transformed into an equivalent semantic-network language that is contained in the special class of category-1 network-languages. Q.E.D.

While this rule-to-network language transformation is quite simple, the transformation of a semantic-network language into a production-rule language is not. The

problem that has to be overcome is the transformation of the relation-words that are not identical to the symbol  $m$ . In a network-to-rule transformation, a one-for-one transformation to the  $m$  symbol is not possible when several relations exist. If a many-to-one transformation is used, then important relationship information is lost.

To keep from losing this relationship information, a fact-word in the production-rule word must exist that contains this relation information as well as the node-label information. That is, when a semantic-network relation-word is not equivalent to  $m$ , the  $a$  or the  $b$  fact-words in the  $amb$  rule-word must contain both the relation and the node-label word. In effect, the fact-alphabet has to become  $Z \cup Y$ . As an example, let "CR3D" represent a semantic-network word where "C" and "D" are specific node-label words and "R3" is a single relation-word that is not equivalent to  $m$  (i.e. C is related to D by R3). Then, one possible transformation of this network-word yields a rule-word of the form "Cm(R3D)." This rule-word reads "C implies is related to D by R3." While somewhat awkward, the same information that is contained in the semantic-network word is also contained in the resulting production-rule word. However, there is a problem with the resulting production-rule language when it is used in a production-rule system.

This problem is that the production-rule language only allows production-rule systems to be constructed in which each of the "(R3D)" fact-words is a terminal condition

(answer). This can be seen from the description of production rules given in Section I. A production rule "fires" when the antecedent (C) is matched from the known information that is contained in the situation data (see Figure 4). The known information is either entered by a user or by the firing of a rule. Because the C position never contains a relation-word, the rule  $Cm(R3D)$  never activates any other rule once it is fired. That is, the "(R3D)" fact-word never occurs in the antecedent position (a) of the amb production-rule word. If another semantic-network word transforms into the production-rule word " $Dm(R4F)$ " (D implies is related to F by R4), then firing rule  $Cm(R3D)$  does not activate the  $Dm(R4F)$  rule as it should. However, there is a way to correct this problem.

To solve this problem, the definition of category-1 semantic-network languages needs to be reviewed (see Theorem 6.15 of Section VI). Each of these network-languages consists of all the words of the form  $arb$  where  $a$  and  $b$  are members of a regular set. That is, words of the form " $CR(i)D$ " and " $DR(i)C$ " exist in each network-language. Each node  $D$  can be reached from node  $C$  by a relation  $R(i)$  and each node  $C$  can be reached from  $D$  by the same set of relations. This cycle exists because the language has to contain all possible words so that users can construct their particular implementations of a semantic-network any way they desire.

Because this cycle results in all node labels being preceded by a relation, using the "[R(i)D]" form as the fact-words preserves the needed relation-information and corrects the "dead-end" problem. That is, every semantic-network word is transformed into a production-rule word of the form "[R(i)C]m[R(j)D]" where "i" and "j" may or may not be equal. If the relation-set contains the implies relation  $R_I$ , the CRID word-forms can translate directly into CmD word-forms. Therefore, the new production-rule language has to consist of words of the following form: CmD, Cm[R(j)D], [R(i)C]mD and [R(i)C]m[R(j)D] in order to account for all possible semantic-network words. Of course, the awkwardness in reading the rule remains and is somewhat worse. But, all information is preserved and the production-rule systems using this language are able to operate as expected.

While these word transforms solve the dead-end problem by using a particular partition of the semantic-network word, they do not represent a legal transformation. As a matter of fact, no legal transformation exists for an arbitrary semantic-network language contained in the special class of category-1 network-languages.

Theorem 7.08: There does not exist a single information-lossless transformation that will convert an arbitrary semantic-network language contained in the special class of category-1 network-languages into a production-rule language contained in the special class of category-1

rule-languages (see Theorem 5.13 of Section V and Theorem 6.15 of Section VI for class definitions).

Proof: This proof follows from the definitions of semantic-network and production-rule languages. While all four word-forms discussed in the paragraphs preceding this theorem need to be available in order to transform any network-word into a rule-word, they have brought out the specific condition that can be used to prove this theorem. For an information-lossless network-to-rule transformation, the resulting fact-set must consist of words where the individual relation/node-label associations (e.g. R3D, R2C) are identifiable. The term "identifiable" is used to mean a one-to-one mapping of the relation/node-label associations to unique words in the fact-set, not necessarily a symbol-by-symbol mapping of the relation/node-label strings to the fact-strings. That is, every fact-word must represent one of the unique relation/node-label associations so that information is not lost in the transformation.

However, the fact-set consisting of these equivalent relation/node-label words is directly affected by the previously identified



AD-A172 516

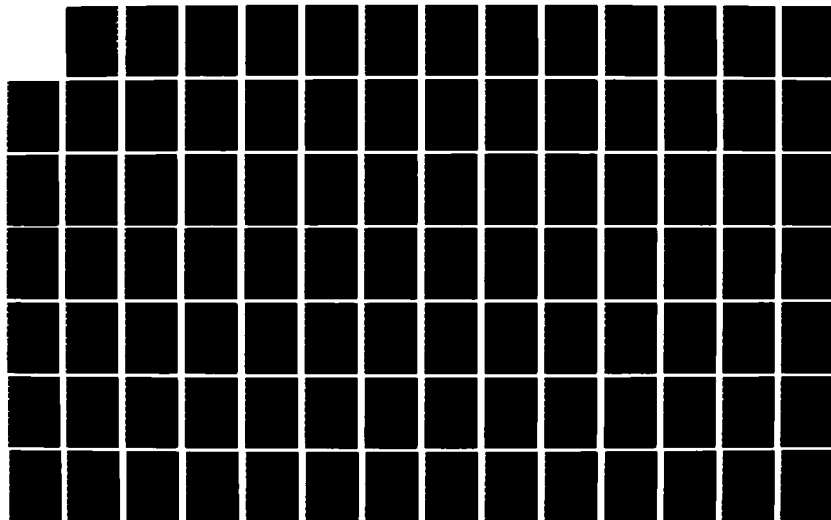
A NON-COGNITIVE FORMAL APPROACH TO KNOWLEDGE  
REPRESENTATION IN ARTIFICIAL (U) AIR FORCE INST OF  
TECH WRIGHT-PATTERSON AFB OH SCHOOL OF ENGI  
J A MCMANNAMA JUN 86 AFIT/DS/ENG/86J-1

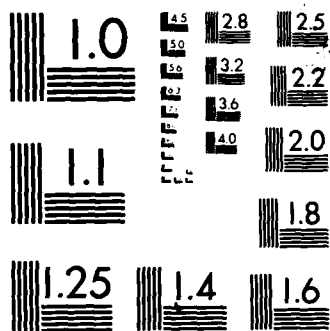
3/4

UNCLASSIFIED

F/G 9/4

NL





MICROCOPY RESOLUTION TEST CHART  
NATIONAL BUREAU OF STANDARDS-1963-A

semantic-network cycles  $CR(i)D/DR(i)C$ .

Because of these cycles, every relation has to be associated with every node label. Hence, this transformed set of facts must contain individual words for every relation/node-label association. For example, let the node-label set contain only two labels, C and D, and the relation-set contain only two relations, R2 and R3. Then, to handle all the associations, the fact-set must contain four words to represent the R2C, R2D, R3C and R3D relation/node-label associations.

This requirement that the fact-set must contain words for every relation/node-label association creates a problem with any network-to-rule transformation. By definition, a production-rule language that is a member of this special category-1 class of rule-languages has to consist of all possible words of the form  $amb$  where  $a \neq b$ . This inequality constraint results in production-rule words existing in the language where the  $a$  and  $b$  fact-words can differ only by the individual symbol placement in the fact-word. That is, if the fact-set contains unique words for every one of the relation/node-label associations, the production-rule language

must then contain rule-words in which the node-label portion of the relation/node-label association is identical in both the a and b fact-words but the relation-portion is different. For example, if the fact-set contains two words representing two relation/node-label associations, R2C and R3C, then, the production-rule language must contain the rule-words representing the equivalent of "R2CmR3C" and "R3CmR2C."

However, by the definition of a semantic-network language, the  $a \neq b$  node-label constraint prevents equivalent types of network-words from occurring. That is, no semantic-network word exists where identical node labels occur in the a and b positions of the arb word-form. Therefore, by applying any general transformation on these network-words, the resulting production-rule words must maintain the unequal node-label constraint. For example, let a semantic network consist of two network-cycles, CR2D/DR2C and CR3D/DR3C, where neither relation is the implies relation. Any information-lossless transformation that maintains the node-label inequality has to generate a set of rule-words that consists of the equivalent rule-words: R2CmR2D,

R2CmR3D, R3CmR2D, R3CmR3D, R2DmR2C, R2DmR3C, R3DmR2C and R3DmR3C. Yet, by the definition of a production-rule language, equivalent rule-words of the form R2CmR3C, R2DmR3D, R3CmR2C and R3DmR2D must also exist in a rule-language over the four equivalent fact-words of R2C, R3C, R2D and R3D.

So, no matter how the relation/node-label associations are partitioned, there must always exist a unique fact-word representing each unique association. Therefore, any general information-lossless transformation used to transform an arbitrary category-1 semantic-network language into a production-rule language actually transforms the network-language into a subset of a production-rule language. This subset of rule-words itself does not meet the formal definition of a category-1 rule-language. By combining this finding with the existence of the special-case transformations of Theorems 7.01 and 7.04, it follows that no single information-lossless transformation exists that will transform an arbitrary semantic-network language contained in the special class of category-1 network-languages into a production-rule language

contained in the special class of category-1 rule-languages. Q.E.D.

This theorem makes a very powerful statement. Even though special cases exist where category-1 semantic-network languages can be transformed into equivalent category-1 production-rule languages, a general transform does not exist for all semantic-network languages contained in this special class of category-1 network-languages. Therefore, anyone performing any category-1 network-to-rule transformation has to prove that their transformation is correct for their special class of languages. Because part of the proof of Theorem 7.08 showed that general network-to-rule transformations do not necessarily generate a true production-rule language, the proof of correctness of special-case transformations may be very involved if not impossible.

Additional affects of this theorem, as well as other theorems contained in this section, upon rule and network languages are discussed in Section IX. Section X contains suggestions regarding the techniques to use in proving the correctness of special-case transformations. However, the specific affects that all the previous theorems have upon the remaining production-rule and semantic-network languages in the definition hierarchy (see Section IV) are given in the next section.

### VIII. Hierarchical Language Results

So far this dissertation has dealt with only a special class of category-1 production-rule and semantic-network languages (see Theorem 5.13 of Section V and Theorem 6.15 of Section VI for class definitions). While detailed analysis of all possible classes of category-1 languages are not performed herein, the results of the analysis of these special classes of rule and network languages are used to determine characteristics of higher category-level representation-languages (see Section IV). From a review of the proofs of these category-1 theorems in Section V and VI, it can be seen that the proofs for semantic-network languages only differ slightly from those of production-rule languages. Therefore, instead of giving separate hierarchical theorems and propositions for each of these representations, each theorem and proposition in this section covers both rule and network representations. To begin developing these hierarchical theorems, a review of the tables in Section IV is required.

#### Hierarchical Language Expansion

After detailed evaluation of the tables in Section IV, it is noticed that the differences among categories of knowledge-words contained in these representations involve only the form used to represent the knowledge. For example, category-1 knowledge-words represent knowledge as simple

strings of symbols while a higher category level uses functions with free variables to represent some of the knowledge. From the investigations of the special category-1 classes of languages, the language-type of the sets of knowledge-words (i.e. the language-types of the fact, node-label and relation sets) was found to affect the overall language-type of the representation.

The most critical impact was identified by the category-1 theorems that ignore the language-type of the fact, node-label and relation sets altogether (see Theorem 5.08 of Section V and Theorem 6.08 of Section VI). These theorems revealed that the resulting overall knowledge representations did not have formal language equivalents at all. Hence, if the complexities of the knowledge-languages are ignored at all the category levels, the same results would be expected to occur. That is, if the language-type of the knowledge-words contained in a representation's structure is ignored, then no formal language equivalents exist for the overall knowledge representation. This leads to the first hierarchical theorem for these two knowledge representations.

Theorem 8.01: Given any set consisting of all possible production-rule or semantic-network strings where the strings have the characteristics listed for one of the category levels of the definition-hierarchy (see Section IV). If the antecedent/consequent (fact), node-label



and relation sets are treated as only symbols in alphabets, then, in general, the resulting overall knowledge representations formed by using these sets of strings are not formal languages.

Proof: This proof follows directly from the category-1 findings. Because the category-1 characteristics are included in all higher numbered categories of characteristics, each of the higher numbered categories of fact, node-label and relation sets has to contain special subsets for the strings that meet the category-1 characteristics. From Definition 5.01 of Section V and Definition 6.01 of Section VI, one of these special category-1 subsets of strings for each of the fact, node-label and relation sets are countably infinite in size. Therefore, by using the higher numbered categories of fact, node-label and relation sets that contain these particular category-1 subsets in the overall languages' alphabets, the size of these alphabets has to be at least countably infinite.

Now, from the grammar definition (Definition 1.02 of Section I), a language-generating grammar has to have a finite alphabet of terminal symbols. Since the alphabets for

these higher numbered categories of representations are at least countably infinite, no grammar exists that will generate formal language versions for these higher numbered categories of representations.

However, if each of the category-1 subsets is finite as well as the number of remaining symbols in the higher category-level alphabets (the knowledge-language could still be ignored even though these finite sets are type-3 languages over a different alphabet), then the grammar exists and a language version also exists. Therefore, depending on whether or not the set of knowledge-words is countably infinite, these production-rule and semantic-network representations cannot be formal languages. Q.E.D.

While Theorem 8.01 proved that cases exist where the production-rule and semantic-network representations cannot be formal languages, it remains an open question as to the exact language-types, if any, of the representations that contain knowledge-words which have all the characteristics listed for the top category of the definition-hierarchy (see Section IV). The problem comes from developing all the detailed constraints needed to generate an all-inclusive description of such a complex set of knowledge-words. Questions continue to come up that have no absolute answers.

For example, how deep can Boolean combinations be nested? Are value assignments (like those used in MYCIN) additive so that the  $a \neq b$  constraint has to be modified? Are free variables used in conjunction with procedure calls for variable passing? Can procedure calls be nested? These are but a sample of the problems that have to be overcome to reach a reasonable definition of the top-category characteristics.

Even though these problems are difficult and have not been resolved sufficiently in this dissertation, the conjecture used herein is that the set of knowledge-words at each category level is a formal language and the overall representation that uses these knowledge-words is also a formal language. This conjecture is supported by the existence of various programming languages like OPS5 and ROSIE (101) that operate with representations containing knowledge-words which have some of these complex characteristics. However, because of the possible countably infinite variations, reaching any final conclusion about the hierarchy of rule and network languages based on so small a sample would be foolish if not ridiculous.

In order to provide as much insight into these knowledge representations as possible, this conjecture is assumed to be true for both the production-rule and semantic-network representations. As a result, the findings on the special category-1 classes of rule and network languages can then be used to reach conclusions about these hierarchical sets of

knowledge representations. Since these hierarchical findings will be based on this conjecture, these findings will be presented as propositions, not theorems. Section IX and X contains further discussion on resolving this conjecture so that these propositions may be converted into theorems.

However, before any propositions can be presented, the conjecture must be formally defined. For clarification, separate conjectures will be given for each representation.

Conjecture 8.01: The allowable sets of all possible

knowledge-words formed at each category level of Tables II and III of Section IV produce formal language equivalents. When each of these knowledge-languages is combined with the production-rule's structure, a formal language equivalent exists for the overall representation.

Conjecture 8.02: The allowable sets of all possible

knowledge-words formed at each category level of Tables IV, V and VI of Section IV produce formal language equivalents. When each of these knowledge-languages is combined with the semantic-network's structure, a formal language equivalent exists for the overall representation.

Using these conjectures, the following propositions can be proven.

Proposition 8.01: Given that Conjectures 8.01 and 8.02 are true, then, in general, the top-category language equivalents of the production-rule and semantic-network representations cannot be context-free (type-2) languages.

Proof: This proof follows by construction and from the formal language evaluations of computer programming languages. By the inclusion hierarchy property of Section IV, the top-category knowledge-languages contained in the production-rule and semantic-network representations have words that allow arbitrary procedure calls to be made. Because these procedures themselves can be implemented in any computer programming language, they may be considered, without loss of generality, as being formal language "sentences" over an alphabet that is disjoint from the rest of the alphabet-symbols in the knowledge-language's alphabet. That is, the procedure-sentence is made up of words that have no common alphabet-symbols with the other knowledge-words.

Now, by construction, remove the procedure-call words from the top-category knowledge-language (a knowledge-language

exists per Conjectures 8.01 and 8.02) and replace them with their equivalent, disjoint sentences. Recall that this dissertation is removing the semantics from the words and evaluating the resulting syntax. Therefore, this replacement action removes the semantics (meaning) of a procedure call by replacing it with the syntax strings that it represented.

Because the procedures are arbitrary, this replacement process will result in incorporating all possible procedure-sentences in order to handle all possible procedures that a user may require. By applying a set's property that duplicate words are not allowed, the union of all the sentences yields all of the words of the original computer programming language that incorporated the procedures. That is, the knowledge-language resulting from this construction contains all the words of the implementing computer programming language as members. Since these words consist of alphabet-symbols that are not used in any other knowledge-word, the words of the computer programming language can be easily identified in the knowledge-

language. Because of this separation of the words of the programming language and the remaining knowledge-words, any machine accepting the knowledge-language will have to accept this implementing computer programming language as a separate unique subset. That is, when words of this subset programming language are encountered, the machine has to use a special set of states and paths that is only used to accept these unique words. In effect, the machine contains a separate acceptor for the computer programming language. Since Harrison has proven several computer programming languages not to be context-free (96:219-221), then, in general, this separate acceptor must be more powerful than a nondeterministic, finite machine with a single pushdown-store, in order to accept the words of the programming language (9:43). Hence, the overall machine accepting the entire knowledge-language must be more powerful than this machine-type as well.

Finally, since this knowledge-language has to be accepted as part of the overall representation language, the disjoint

property of the subset computer programming language also forces the machine accepting the overall representation to be more powerful than a type-2 acceptor. Hence, in general, these top-category production-rule and semantic-network representation languages cannot be type 2. Q.E.D.

The finding of Proposition 8.01 has to be general because only several but not all computer programming languages have been evaluated. However, there is another case that can support the general nature of the finding. It may be the case that only certain types of procedures are allowed. Then, the entire computer programming language may not be a subset language. For these cases, the trace language techniques would have to be used to make the determination (88). But, knowing that the top-category languages have a good chance of being type 1 or type 0 indicates that the detailed formal language analysis of the top-category representations can become very complex.

Even the general finding of Proposition 8.01 still points out once again that a representation's structure alone does not control the final language-type. The form that the knowledge-language itself takes (procedural vs. declarative), greatly affects the final language equivalent of the specific representation. With this lower bound of type-1 languages established for the general case, the



investigation now shifts from the individual languages to the classes of languages formed at higher category levels.

Proposition 8.02: Given that Conjectures 8.01 and 8.02 are true, then the entire class of production-rule languages and the entire class of semantic-network languages formed at any of their respective category levels (see Section IV) is not closed under the union operation.

**Proof:** This proof follows by construction of a counter-example. For the production-rule representation, select the same two finite languages that were used in the proof of Theorem 5.15 of Section V. For the semantic-network representation select the same two finite languages that were used in the proof of Theorem 6.17 of Section VI.

Now, consider the entire category-1 set of rule (network) languages first. Since each of the special classes of category-1 languages consists of all the possible category-1 finite languages for a given representation, it follows from Theorem 5.15 (Theorem 6.17) that the third finite language resulting from the union of two previously selected finite rule

(network) languages cannot be contained in any of the remaining classes of category-1 rule (network) languages. Because this third finite language is not a finite rule (network) language, it follows that the the entire set of all category-1 production-rule (semantic-network) languages is not closed under union.

Next, consider the remaining category-levels of classes of rule and network languages. The inclusion hierarchy property (see Section IV) requires that every rule (network) language contain a subset of words which have category-1 characteristics. Conjectures 8.01 (8.02) requires that this subset be a formal language. Because of these requirements, there exist in every class of higher category-level rule (network) languages at least two separate languages that contain, as a subset, one of the two specific finite category-1 rule (network) languages used in Theorem 5.15 (Theorem 6.17). That is, there exist two production-rule (semantic-network) languages in each category-level class of rule (network)

languages that contains these finite subsets as their only category-1 language.

Now, perform the union of these two higher category-level production-rule (semantic-network) languages. This union produces a third set of rule (network) words that contains the third finite rule (network) language as the category-1 word-set. However, from the proof of Theorem 5.15 (Theorem 6.17), the third finite language is not a production-rule (semantic-network) language. From the inclusion hierarchy property and Conjectures 8.01 and 8.02, this third higher category-level rule (network) language cannot be contained in the higher category-level class of rule (network) languages, since it does not contain a category-1 rule (network) language as a subset. Hence, the entire class of production-rule (semantic-network) languages at any category level is not closed under the union operation. Q.E.D.

The union closure property is not the only property that these classes of languages do not meet. These languages do not meet the product closure property.

Proposition 8.03: Given that Conjectures 8.01 and 8.02 are true, the entire class of production-rule languages and the entire class of semantic-network languages formed at any of their respective category levels (see Section IV) are not closed under the product operation.

Proof: This proof is by construction. The inclusion hierarchy property (see Section IV) requires that every representation-language contain a subset of words which have category-1 characteristics. Conjectures 8.01 and 8.02 require that this subset of words be a formal language. Because of these requirements, every higher category-level representation-language must contain all the words of some category-1 representation-language as a subset. From Definitions 4.01 and 4.02 of Section IV, the form of the words in the category-1 language subsets are  $amb$  and  $arb$ , respectively.

Now, perform the product operation on any two production-rule (semantic-network) languages in the same category-level class. The resulting set of rule (network) strings contains category-1

level strings of the form "AlmBlA2mB2" (AlR2BlA1R3B2). That is, the category-1 rule (network) strings contain multiple m (relation) symbols (words). There no longer exist any of the required category-1 words of the form amb (arb) in the language. Therefore, from the inclusion hierarchy property and Conjectures 8.01 and 8.02, the rule (network) language formed after the product operation cannot be contained in the entire class of production-rule (semantic-network) languages at that category level. Hence, the entire class of production-rule (semantic-network) languages at any category level is not closed under the product operation.

Q.E.D.

The fact that no entire class of production-rule or semantic-network languages is closed under either the union or product operations at any category level leads to the last proposition on the individual characteristics of these classes of languages.

Proposition 8.04: Given that Conjectures 8.01 and 8.02 are true, then the entire class of production-rule languages and the entire class of semantic-network languages formed

at any of their respective category levels (see Section IV) are not Abstract Families of Languages (AFL) (24).

Proof: From reference 24, an Abstract Family of Languages (AFL) contains languages closed under six operations: union, product, Kleene closure, e-free homomorphisms and intersection with a regular set. From Propositions 8.02 and 8.03, the entire class of production-rule (semantic-network) languages is not closed under either union or product at any category level. Therefore, the entire class of production-rule (semantic-network) languages at any category level is not an AFL. Q.E.D.

As was the case for similar findings on the special classes of category-1 rule and network languages, the results of Proposition 8.04 present a major stumbling block to the study of these entire classes of production-rule and semantic-network languages. Since these classes are not AFLs, the determination of additional characteristics of these classes has to involve analysis of the entire set of languages contained in each class. With all the various combinations of possible languages, each category level can contain countably infinite classes each of which contains countably infinite languages. With the loss of the

multitude of characteristics proved about AFLs by formal language theorists, developing a small subset of additional characteristics for these classes of representations can be very time consuming. From Godel's theorem (89), it may be the case that all the language characteristics of these two representations can never be determined.

Even if this is the case, there are still additional characteristics about these hierarchical classes of languages that deserve further attention. One of these characteristics is the equivalence between a top-category production-rule language and a top-category semantic-network language.

#### Hierarchical Equivalences

A detailed review of the equivalence theorems of Section VII, shows that the reason that equivalence exists between certain production-rule and semantic-network languages is that isomorphic fact and node-label sets could be constructed and the bijective  $R1=m/m=R1$  mapping could be used. These isomorphic mappings are possible because the needed detailed descriptions of the facts, node-label and relation sets are provided in the category-1 definitions of Section IV (see Definitions 4.01 and 4.02). For the higher level cases, these detailed definitions have not been provided.

Because these detailed definitions have not been and may never be (see Section IX for discussion) sufficiently developed, an additional conjecture is needed to be able to

determine the equivalence between higher category-level rule and network languages. From the detailed literature review performed to develop the tables in Section IV, the descriptions of the antecedents/consequents (facts), node-labels and relations appear to agree, in general, at various levels. For example, both the rule's antecedents/consequents and the network's node labels can contain procedure calls, free variables, etc. However, these equivalent descriptions do not necessarily occur at the same category level number in both representations.

But, if all the characteristics are considered (i.e. the top-category definition), then all the production-rule's characteristics appear to have general agreement with some of the semantic-network's characteristics. The word "some" is used because it appears that a few of the network's characteristics do not have a direct rule equivalent. For example, the network's exception-links do not directly match any of the rule's characteristics.

Because this equivalence of characteristics appears to be complete for the rules, the conjecture used in this dissertation is that all of the top-category rule-characteristics are indeed equivalent to some of the top-category network-characteristics. A formal definition of this conjecture follows. Section IX contains further discussion on resolving this conjecture so that the following propositions involving it may be converted into



theorems. Section X contains suggestions on how to develop special cases that support this conjecture.

Conjecture 8.03: All of the general descriptions for the production-rule characteristics given in Tables II and III of Section IV are identical to some of the general descriptions for the semantic-network characteristics given in Tables IV, V and VI of Section IV.

With this third conjecture, the equivalence of the production-rule and semantic-network language can now be determined. The equivalence investigation begins by determining the equivalence of the single top-category rule and network language that consists of all possible words.

Proposition 8.05: Given that Conjectures 8.01, 8.02 and 8.03 are true, then the single top-category production-rule language consisting of all the possible rule-words is not equivalent to the single top-category semantic-network language consisting of all the possible network-words.

Proof: This proof follows from a category-1 equivalence theorem, the inclusion hierarchy property of Section IV and Conjectures 8.01, 8.02 and 8.03. Because this theorem requires the top-category languages to consist of all words, it

follows from the inclusion hierarchy property and Conjectures 8.01 and 8.02, that each of these top-category languages must contain the entire set of their respective category-1 representation-words as a subset category-1 language. These special all-inclusive category-1 word-sets were evaluated in Sections V, VI and VII as the PR production-rule language and the SN semantic-network language. From these evaluations of PR and Sn, Theorem 7.03 of Section VII established that PR is not equivalent to SN because SN contains more words than PR.

Excluding the PR and SN subsets for the moment, by Conjecture 8.03, the remaining words in the top-category production-rule language are equivalent to some of the remaining words in the top-category semantic-network language. That is, because the characteristics in the tables of Section IV are incorporated as words in these representation languages, some of the remaining semantic-network words must be equivalent to those in the remaining set of production-rule words.

However, there are more remaining network-words than rule-words.

Since both the SN subset and the remaining semantic-network words outnumber the PR subset and remaining rule-words, the top-category all-inclusive rule-language cannot be equivalent to the top-category all-inclusive semantic-network language. Q.E.D.

Even though the single top-category languages are not equivalent, various subsets of top-category level rule and network languages can be shown to exist that are and are not equivalent to each other. Because of the fact that the characteristics at a specific lower category-level of rule-languages do not match the semantic-network characteristics at the same category-level number, these subsets of languages can only be proven to exist at the top category. While proving that these subsets exist provides additional information about the top-category languages, all of these subsets are not needed to prove the final equivalence proposition. If needed, the reader can prove that these subsets exist by using the proof technique used in the following proposition and substituting the specific category-1 theorem from Section VII for Theorem 7.06. For example, by substituting Theorem 7.05 for Theorem 7.06, a set of countably infinite top-category network-languages can

be proven to exist that are not equivalent to any production-rule language.

Proposition 8.06: Given that Conjectures 8.01, 8.02 and 8.03 are true, then the entire class of top-category production-rule languages is not equivalent to the entire class of top-category semantic-network languages.

Proof: This theorem follows from Theorem 7.06 of Section VII, the inclusion hierarchy property of Section IV and the three given conjectures. Theorem 7.06 of Section VII established that the special classes of category-1 production-rule and semantic-network languages are not equivalent (see Theorem 5.13 of Section V and Theorem 6.15 of Section VI for class definitions). This theorem was proved by showing that there exist both finite and countably infinite semantic-network languages that contain more words than any finite or countably infinite production-rule language. This proposition can be proven using this same unequal-size characteristic.

First, the inclusion hierarchy property requires that every representation-language contains a subset of words which have category-1 character-

istics. Conjectures 8.01 and 8.02 require that this subset of words be a formal language. Because of these requirements, there exist top-category rule and network languages that contain all the words of one of their respective category-1 languages as a subset. That is, each one of the top-category languages must contain a category-1 language as a subset. Since the languages contained in these special classes of category-1 languages qualify as one of these subset languages, there must exist at least one top-category language for every one of these special category-1 languages. Because of the large number of word-sets that can have higher category-level characteristics, more than one top-category language can exist that contains the same special category-1 language as a subset. However, the maximum number of such languages is bounded at countably infinite by the language-space bound and Theorem 3.05 of Section III.

For the moment, exclude these special category-1 subsets from the the top-category languages. By Conjecture 8.03

and the fact that each of these top-category rule and network languages have to contain all possible representation-words given a particular knowledge-language, there must exist network-languages whose number of remaining words exceeds the number of words remaining for any rule-language. That is, because the characteristics in the tables of Section IV are incorporated as knowledge-words in a representation-language, the "some" stipulation of Conjecture 8.03 means that more knowledge-words exist for the semantic-network languages. Therefore, some semantic-network languages will contain more higher category-level words than any rule-language.

Finally, recombine these remaining subsets of higher category-level words with the special category-1 language subsets. Then, it follows from the unequal sizes of the special category-1 languages and the unequal sizes of these higher category-level word-subsets that there must exist certain top-category network-languages that contain more words than any production-rule language. Hence,

the entire class of top-category  
production-rule languages cannot be  
equivalent to the entire class of top-  
category semantic-network languages.

Q.E.D.

Proposition 8.06 could have been proven directly from Proposition 8.05 by showing the inclusion of the single all-inclusive languages in the top-category classes. However, by using this proof technique, it is revealed that more than one case exists where languages are not equivalent. This existence was shown without having to develop equivalent propositions for all of the special-case theorems of Section VII (e.g. Theorem 7.05). Further impacts of this finding are discussed in Section IX. Section X contains suggestions for knowledge representation researchers to follow when dealing with equivalence issues.

Even though these top-category classes of knowledge representations are not equivalent, transformation from a top-category production-rule language to a top-category semantic-network language can be made.

#### Hierarchical Transformations

As was the case in Section VII, there are two types of transformations that need to be considered: rule-to-network and network-to-rule. The rule-to-network is investigated first.

As previously indicated, the top-category production-rule to top-category semantic-network language transformation does exist. Because the characteristics of the rules at the individual category levels do not match the characteristics of the networks at the same category-level number, the transformation can only be proven to exist for the top category. Also, because the definitions of the characteristics are not provided, detailed definition of the transformation steps cannot be given. However, if Conjecture 8.03 is true, then the transformation for the characteristics does exist. By assuming Conjecture 8.03 is true, the following proposition regarding this rule-to-network transformation is proven.

Proposition 8.07: Given that Conjectures 8.01, 8.02 and 8.03 are true, then any top-category production-rule language can be transformed into an equivalent semantic-network language.

**Proof:** This proof follows directly from Theorem 7.07 of Section VII, the hierarchy inclusion property of Section IV and the three given conjectures. By the inclusion hierarchy and Conjectures 8.01 and 8.02, various category-1 languages exist as subsets of the top-category representation-languages. However, the words in these subset languages all



come from the category-1 word-set. Theorem 7.07 established that an equivalent rule-to-network transformation was possible for the special classes of category-1 languages (see Theorem 5.13 of Section V and Theorem 6.15 of Section VI for class definitions). Since the set of all category-1 words is included in these special classes (PR and SN), the transformation of Theorem 7.07 of Section VII can be used to transform all the category-1 words. Therefore, the transformation given in Theorem 7.07 becomes a subset of the overall top-category rule-to-network transformation.

To handle the remaining words in the top-category rule-languages, Conjecture 8.03 must be invoked to establish that every top-category production-rule characteristic has an equivalent form in the set of semantic-network characteristics. That is, a transformation exists for the rule-words incorporating these characteristics into equivalent semantic-network words that incorporates these same characteristics. Therefore, by combining the category-1 transformation with the trans-

formation for these equivalent characteristics, any top-category production-rule language can be transformed into an equivalent semantic-network language.

Q.E.D.

While this production-rule to semantic-network transformation relies almost entirely on the acceptance of a conjecture, the nonexistence a single semantic-network to production-rule transformation can be proven from the category-1 results.

Proposition 8.08: Given that Conjectures 8.01, 8.02 and 8.03 are true, then there does not exist a single information-lossless transformation that converts an arbitrary top-category semantic-network language into a top-category production-rule language.

**Proof:** This proof follows directly from Theorem 7.08 of Section VII, the inclusion hierarchy property of Section IV and the three given conjectures. Theorem 7.08 proved that no information-lossless transformation exists for the special classes of category-1 languages (see Theorem 5.13 and Theorem 6.15 for class definitions). The reason was that the transformed rule-grammar would generate category-1 rule-words that have no

relation to the original semantic network. Hence, any special category-1 network-to-rule transformation would result in a set of rule-words that was not a proper production-rule language.

While Theorem 7.07 only dealt with a special class of category-1 languages, the results actually hold for any category-1 language. The problem with the transform involved the internal words of the category-1 representation-words. Since all category-1 words have the same form and meet the  $a \neq b$  constraint, any category-1 production-rule grammar will, in general, still generate rule-words that contain the same node label on both sides of the  $m$  symbol. Hence, the violation of the  $a \neq b$  constraint on the network-words still occurs.

Recognizing that this violation holds for all category-1 languages, consider the top-category languages. From the inclusion hierarchy property and Conjectures 8.01 and 8.02, it follows that each of the top-category languages is formed by adding more words to some category-1 language. That is, every top-category production-

rule language and every top-category semantic-network language contains all the words of a category-1 language as a subset. Hence, any top-category network-to-rule transformation has to transform the category-1 network-language into a set of rule-words.

Now, consider the only three forms that this transformation of the category-1 subset language can take. The first form involves transforming all the category-1 network-words into higher category-level rule-words. That is, the resulting set of rule-words does not contain any category-1 words. This transformation is illegal because, from Conjectures 8.01 and 8.02, the production-rule language has to contain a nonempty subset of category-1 words.

The second form involves transforming the category-1 network-language into a category-1 rule-language. This is also illegal, in general, from the previously mentioned expansion of the results of Theorem 7.07 to include the entire set of category-1 languages.

The third form involves transforming part of the category-1 semantic-network words into higher category-level rule-words and the remaining category-1 network-words into category-1 rule-words. There are two conditions on the form of this transformation that have to be considered. The first condition is that the resulting set of category-1 rule-words is not a complete production-rule language. For this condition, the transformation is illegal since the rule-language must contain all the words of a complete category-1 rule-language as a subset.

The second condition is that the resulting set of category-1 rule-words does form a category-1 rule-language. This transformation is also illegal. This is because the transformation results in mapping some of the knowledge-words (see Definitions 4.03 and 4.04 of Section IV) contained in the overall category-1 network-language, into two knowledge-words contained in the knowledge-language for the overall rule-representation. These two knowledge-words are actually at different category levels of definition.

In effect, the resulting rule-language has added characteristics to the semantic-network knowledge-words that did not originally exist. That is, the rule-language has gained knowledge. For example, the category-1 relation/node-label association R2C transforms into a category-1 fact-word, R2C, and a predicate-word "R2C(x,x)."

To understand why this problem occurs, the semantic-network relation/node-label cycle needs to be reviewed. No matter how the category-1 network-language subset is partitioned, the network-cycle CR2D/DR2C forces some node-labels and/or relations to be duplicated in both partitions. For these duplicated knowledge-words, the information-lossless requirement forces this transformation to convert these words into an equivalent category-1 knowledge-word and some higher category-level knowledge-word. Hence, the original category-1 network-words are transformed into two different category-level words.

To better understand how this effect-ively adds a characteristic to the semantic-network language, consider the

previous example's knowledge-words  $R2C$  and  $R2C(x,x)$ . Make use of the fact that a rule-to-network transformation exists and transform these equivalent knowledge-words back into their network word-forms. Because the node label and relation words are over disjoint alphabets, let the rule-to-network transformation use a relation/node-label separator to retrieve the node labels and relations. Using this rule-to-network transformation, it then follows from Proposition 8.07 that the category-1 word  $R2C$  transforms into node labels for the network. Applying the separator, the actual node label and relation words can be obtained. Hence,  $R2C$  transforms into  $R2$  and  $C$ .

However, for the  $R2C(x,x)$  knowledge-word, it follows from Conjecture 8.03 and Section IV that network-words exist whose node labels can contain predicates. Depending on the definition of  $n$ -ary relations, the relation-set may also contain predicates. Therefore, applying the transformation steps for these predicate-words and the separator, a new predicate node-label " $C(x,x)$ " and a new

predicate relation "R2(x,x)" are added to the network that did not originally exist.

Now, it may be argued that these R2C(x,x) words are syntactically different from R2C words and could be identified and mapped so that no extra words are added to the network. While this depends on whether or not the R2C(x,x) words form a language by themselves (it takes a Turing acceptor to identify these words), the point is that the production system using this equivalent production-rule knowledge representation may reach conclusions that could never be reached by the original network-system. The added characteristics actually reside inside the production-rule representation.

Since all the possible transformation-forms have been shown to be illegal for the general case, it follows that no single information-lossless transformation exists that will transform an arbitrary top-category semantic-network language into a top-category production-rule language. Q.E.D.

The finding of Proposition 8.08 may be hard for some people to accept. In one case, there are those who may



argue that embedded procedures can be used to handle the problems of the transformation. This type of statement can be seen to be incorrect by realizing that a transformation is nothing more than a procedure itself. Since a single one does not exist, the embedded procedure-calls can only work for a limited number of cases (if any).

In another case, there are those who may argue that the hierarchical approach is not the only way to define a representation's characteristics. Using other definition approaches, the inclusion constraint can be removed from the proof of Proposition 8.08 and a transform may be able to be found. This argument has merit in that special cases may be found where a transformation does exist (see Theorems 7.02 and 7.05). However, this transformation is good for only that special case or some special subset of cases. As soon as their languages contain all of the characteristics of Section IV (if this is assumed to be a complete list), then they always have other language versions where the transform does not work (e.g. the top-category languages of Proposition 8.08). Because of the countably infinite variations possible, every special-case transformation has to be proven to operate correctly on the specific class of languages before it can be used in a application.

While other arguments may exist, the findings of this section still remain sound under some of the more severe complaints. Because they are sound, no further analyses of these hierarchies of representation-languages are performed

in this dissertation. Therefore, this concludes the formal analysis of production-rule and semantic-network languages. However, there are some important conclusions that can be reached regarding the affects that the findings of Sections V-VIII have on the field of Artificial Intelligence Knowledge Representation.

## IX. Conclusions and Discussion

### Conclusions

The analyses of production rules and semantic networks via formal language theory provide conclusions that affect the knowledge representations. The first conclusion is that both the knowledge-language (see Definitions 4.03 and 4.04 of Section IV) and the representation's structure must always be included during formal language analyses of knowledge representations. The rationale for this conclusion follows.

Theorems 5.01 through 5.06 and 6.01 through 6.06 of Sections V and VI, respectively, revealed that even though the knowledge-language is a regular set, the constraints of the representation's structure causes the final language not to be regular. On the other hand, if the knowledge-language type is ignored as in Theorems 5.07, 5.08 and Theorems 6.07, 6.08 of Section V and VI, respectively, the overall set of representation-words fails to include formal words. So, no formal language versions exist for these types of knowledge representations.

Even when these types of representations are forced to be languages (see Theorem 5.12 of Section V and Theorem 6.14 of Section VI), the resulting finite languages are unable to handle all the possible representation-words. Haines even conjectured that his type-2 language finding for his

centermarker-languages depends on the regular set constraint of the language used for the x's and y's in his "xcy" words (98). Therefore, the representation's structure and knowledge-language must be considered together when evaluating knowledge representations.

Another conclusion is that for either a production-rule or semantic-network representation, no single support-tool can be designed to handle all possible knowledge-languages. This conclusion has considerable impacts on the selection of knowledge-base support-tools, especially in regards to present manufacturers' claims of having developed generic representation support-tools (101). This conclusion is justified in a straightforward manner.

Because knowledge representation researchers cannot agree on any single definition of knowledge (84), the number of the possible knowledge-languages cannot be purposely restricted. This is also true for the number of the resulting overall knowledge representation languages. For example, recalling from Section IV that predicates are allowable knowledge-words (see Definitions 4.03 and 4.04 of Section IV), the set of knowledge-languages could be selected to be the finite set of predicate calculus languages given by Manna (9:Chapter 2). It follows from the language-space bound and set theory that there still exists a countably infinite set of potential knowledge-languages that are not contained within this predicate calculus language set.

Even recognizing this need to account for all possible knowledge-languages, the manufacturer constructing a support tool for representations is only able to select a finite number of knowledge-languages for the tool to handle. From set theory, this still leaves a countably infinite number of possible knowledge languages that cannot be accommodated. The finite set criterion comes from the fact that no known physical machine can store and search for a countably infinite set of language acceptance/generation rules. Hence, no single tool exists that can support acceptance/generation of a production-rule or semantic-network knowledge representation for all possible knowledge-languages.

While the evidence developed in this dissertation supports the nonexistence of this general support tool, it appears to be countered by the existence of support tools that are proposed to handle any knowledge-language (101). It is important to notice that these types of tools only aid in the construction of a syntax representation from user supplied knowledge-words. The manufacturers of these tools have left it up to the user to ensure that the knowledge-words input are members of the particular knowledge-language of interest. Because of the possibility that a particular knowledge-language contains a countably infinite number of words, a machine that accepts this knowledge-language is needed to perform the missing acceptance test for the user. So, while these representation support-tools can handle any knowledge-language, the manufacturers are really misleading

the users into thinking that only one tool is required to support a given representation. However, a countably infinite set of additional language-acceptors are actually needed to accept all knowledge-languages.

Support tools for construction of knowledge representations are not the only type of tools affected by the findings of this investigation. Based on these findings, an additional conclusion is that by using either the production-rule or semantic-network representation as the main data structure, no single automatic-programming tool can be designed that handles all the possible knowledge-languages. In order to understand the justification for this conclusion, a partial description of automatic-programming tools that use knowledge representations is presented first.

Automatic-programming tools make use of a (natural) language parser to extract the necessary data and algorithm information from a specified text (102). For the data information case, the automatic-programming tool also selects data structures to store the data information that was parsed. If these structures are knowledge representations, then the data extracted by the parser are the knowledge-words. Since a parser can only handle the "sentences" of one language, the knowledge-words extracted must come from a single knowledge-language contained within the parent text-language. Because a physical machine can only store and execute a finite number of parsers, it then

follows that a given automatic-programming tool can only parse a finite number of parent text-languages.

With this understanding of the tool, the justification for the conclusion now can be presented. From Section III, there exists a countably infinite number of possible languages. Contained in this set are the parent text-languages that automatic-programming tools can handle. From the category-1 findings alone, there are at least a countably infinite number of unique knowledge-languages (the regular finite sets). So, there is at least a countably infinite number of possible parent text-languages. Since an automatic-programming tool can only parse a finite number of the countably infinite text-languages, it then follows that no automatic-programming tool can be designed to handle all the possible knowledge-languages.

The previous conclusions concerning support-tools and automatic-programming tools seems to indicate limitations on the applicability of such tools. However, this is not necessarily the case. Any physical machine requires that the implementable version of a knowledge representation contain a finite number of words. For the cases where the knowledge-language contains countably infinite words, there exists a countably infinite number of unique, finite subsets of words (see Theorem 3.10 of Section III). Hence, any support/automatic-programming tool that uses a knowledge-language of countably infinite cardinality can generate a countably infinite number of implementable knowledge repre-

sentations. For the case in which the knowledge-language is some natural language, these implementation versions alone may be sufficient.

Since each implementable knowledge representation contains a finite number of words, a conclusion regarding a finite centralized database of knowledge can be stated. The conclusion is that for a finite centralized database, the semantic-network structure is a viable candidate for the storage-form. As will be shown later, the finite restriction comes from the search required to transform a portion of the semantic network into production rules. The term "viable" is used in the conclusion because the semantic network is a specialized case of the frame representation system (see Sections I and II). Without analysis of frames, no final selection of a structure should be made. Before presenting the justification for this conclusion, a short description of the concept of a centralized database is presented.

The concept of a centralized database of knowledge is to store and maintain all knowledge-words of a given knowledge-language in a single representation. Then, when needed, the knowledge representation in the database is transformed into the knowledge representation required for a given application. Since implementable knowledge representations contain a finite number of words, the finite database is capable of storing implementable versions.



With this conceptual understanding of a centralized database of knowledge, the justification for this conclusion can now be presented. This conclusion is justified from Theorem 7.07 of Section VII and Propositions 8.05 and 8.07 of Section VIII. From Theorem 7.07, every production-rule language in the special class of category-1 languages (see Theorem 5.15 for class definition) can be transformed into an equivalent semantic-network language. It is important to notice that the transformation algorithm only uses one semantic-network relation. Thus, the semantic-network language that is produced by transforming any of these production-rule languages is a proper subset of a semantic-network language that contains more than one relation.

Using this subset property, the hierarchy inclusion property of Section IV and Propositions 8.05 and 8.07, it follows that any production-rule language can be transformed into an equivalent semantic-network language. This language is a subset of the all-inclusive top-category semantic-network language. So, any production-rule language can be explicitly stored within this top-category semantic-network language. The reverse transformation (i.e. generating the production rules) is where the finite size constraint becomes important.

To obtain a production-rule language from this top-category semantic-network language, the algorithm of Proposition 8.07 is applied in reverse to a special subset of semantic-network words. This subset of words is found by

locating all of the semantic-network words where the implies relation exists or where the bijective mapping  $R1=m/m=R1$  is allowed. Because this is a search process, the semantic network has to be finite otherwise the search may never terminate. Since the subset of words obtained by this search is also finite, the production-rule language resulting from the reverse transformation of Proposition 8.07 is also finite.

Even though this reverse transformation can be used to obtain some finite production-rule languages from a finite semantic-network oriented database, it can be argued that the finite constraint limits the usefulness of the centralized database. That is, no single, finite, representation-language can contain all the knowledge-words for countably infinite knowledge-languages. While the argument is valid, the physical constraints of finite memory require a finite-size constraint on any database. The key is to recognize that the transformation algorithm does not depend specifically on the knowledge-language. It only depends on being able to use equivalent knowledge-languages for both representations. Therefore, as the finite number of words in the knowledge-language increases, the same algorithm can still be applied to obtain the rule-languages. This results in fewer modifications being made to the database-handling routines as the knowledge increases.

However, the concept of the centralized database is to store knowledge in a single form so that only one form has

to be maintained. The application versions used in knowledge-bases (see Figure 4) are generated from that form. The centralized database itself does not have to be an application version. But, it has to be able to be transformable into application versions. The key is that while in the semantic-network form, maintenance actions on the database can be performed and validated by using a limited set of proven algorithms. Hence, a set of maintenance algorithms for every representation-form does not have to be verified. The verification of algorithms can be very difficult (9).

While a semantic-network form is a viable candidate for a centralized database, it is concluded that a production-rule form is not an acceptable candidate. This conclusion follows from the fact that, in general, production rules do not contain sufficient numbers of words to store a semantic network.

To understand why production rules cannot store a semantic network, the equivalence and transformation characteristics of the category-1 languages need to be reviewed (see Section VII). The PR and SN languages are not equivalent because SN contains more words than PR. In order to store these remaining semantic-network words in a production-rule form, the fact-words in the rule-set would have to involve some form of relation-word mappings.

However, the proof of the nonexistence of a single network-to-rule transformation (Theorem 7.08 of Section VII)

revealed that problems exist when using relation-words in the fact-set. In this transformation proof, by using fact-words containing relation-words, the resulting production-rule language was shown actually to contain rule-words that have no semantic-network word counterparts. That is, when applying the transformation of Theorem 7.07 of Section VII on a production-rule language modified to contain relations, illegal semantic-network words are produced. Since Proposition 8.08 of Section VIII indicated the same transformation problem exists for any category-level of languages, it follows that the production-rule structure cannot be used as the only storage-structure in a centralized database.

#### Discussion

The formal language findings contained herein may affect human modeling. As pointed out in Section VIII, the exact language-type of the top-category production-rule and semantic-network languages remains open. This open problem was observed to stem from the inability of the knowledge representation researchers to agree on the characteristics of a human knowledge-model (84). It may be the case that these disagreements can never be resolved.

This conjecture comes from the proposition that humans can only be proven to be a formal system from a meta-level that humans cannot directly observe (see the syntax/semantic level discussion in Section I and the meta-level discussions

of Hofstadter (89)). That is, humans would have to leave their present level of awareness in order to be able to describe the form of knowledge that they use. Recognizing the impossibility of this task, the meta-level theory leads to the conjecture that there can never be a formal description of knowledge that can be used to finish the hierarchical language-development. Therefore, the language-typing problem as well as the human storage-model issue could remain unsolved for all time.

This meta-level conjecture is not the only reason that the language-typing problem could remain open. Godel established that it is not possible to prove all truths about any formal system equivalent to or more complicated than arithmetic (89). Because languages are members of a formal system and it takes a Turing machine to generate the type-0 languages as well as solutions to arithmetic problems, Godel's theorem yields the possibility that no formal language versions can ever be proven to exist for the remaining languages in the hierarchy. Hence, the language-typing problem would remain open.

This language-typing problem not only exists for the two knowledge representations analyzed herein, but also exists for all forms of knowledge representations. That is, those proponents of natural knowledge representations (103), hierarchical cortical plane knowledge representations (104), combined knowledge representations (see Section I), etc., all have to face a similar open language-typing problem

associated with their representation-forms. This is due to the fact that no matter how complex their representations are, each of their representations has to have a formal language equivalent.

The reason all these knowledge representations must have a formal language equivalent comes from the work of Chomsky, Turing, and others (8; 89; Section III references; Appendix D references). They proved that computers can only accept, generate, or accept and generate formal languages. Since all of these knowledge representations are eventually implemented on a computer system and used as input data (the language being accepted) for a computer algorithm, it follows that they must be formal languages.

However, because the proponents of all of these knowledge representations do not have a widely accepted definition of their representations, any formal language analysis of these representations would eventually reach the same language-definition plateau as was reached in this dissertation. Once again, Godel's theorem and/or the meta-level conjecture would force the open language-typing problem to show up for each of the knowledge representations.

Not only does the potential exist for this language-typing problem to remain open for all knowledge representations, the potential also exists that a complete human computer-model will never be found. As was the case for the knowledge representations, any human model will be implemented on a computer. Since computers only accept/generate

formal languages, they are a form of a formal system. Godel's theorem can then be applied to the model's output language to reach the conclusion that it is possible that no human computer-model can ever be proven to represent the true human being. In the worst case, the meta-level conjecture can be applied to reach the conclusion that no true human computer-model will ever be found.

Even though all of these conclusions appear to complete the formal language analysis on knowledge representation, there are still recommendations that can be made for additional work.

## X. Recommendations and Closing

### Recommendations

With the success that was achieved by applying formal language techniques to the production-rule and semantic-network knowledge representations, there are several recommendations that can be made. One recommendation is to continue to apply formal language theory techniques to the special classes of category-1 production-rule and semantic-network languages (see Theorem 5.13 of Section V and Theorem 6.15 of Section VI for class definitions), to determine if other special characteristics exist for these languages over and above those known for type-2 languages. By recognizing the existence of these special characteristics, other general conclusions about the top-category knowledge representations may be found. For example, while type-2 languages are closed under union and product, proofs were provided in Sections V and VI that these special classes of category-1 languages are not closed under either union or product operations. As a result, these special classes of category-1 languages as well as the classes of top-category languages were proven not to be an Abstract Family of Languages (AFL) (24). This finding prevented the application of the well published characteristics of AFLs to these knowledge representations.



Another recommendation is to investigate the characteristics of each of the special classes of category-1 production-rule and semantic-network languages formed when the knowledge-language (see Definitions 4.03 and 4.04 of Section IV) is one of three other language-types: type 0, type 1 and type 2 (8). This dissertation only covered the category-1 languages formed from a type-3 knowledge-language. Since Haines conjectured that his type 2 finding for his centermarker-languages depended on the regular set membership of his equivalent knowledge-language (98), a different language classification may be found for these representations when using these other types of knowledge-languages. By remaining at the category-1 level, the complexities of the higher category-level knowledge-words (see Definition 4.03 and 4.04 of Section IV) can be avoided. This should make the task of determining the language-type of the related representation-languages somewhat easier.

A more general recommendation is to apply the formal language techniques of this dissertation to the other forms of knowledge representations. By using the inclusion-hierarchy definition-approach, basic formal knowledge can be gained about the other representations even though the language-typing problem for the top-category languages remains open for them. This new information combined with that of this dissertation can be used to establish foundations from which new knowledge-representation forms can be developed for specific applications. While this

recommendation is to study all knowledge representations, a specific knowledge representation should be evaluated first.

The specific representation is the frame. In Section IX, it is concluded that a semantic-network form could be used as the storage-structure for a centralized database. However, because the network is a special case of a frame representation (the network-hierarchy is contained as a subset of the frame-hierarchy, see Sections I and II), it may be the case that a frame is the best storage-structure for a centralized database. Of course, there are still a lot of other knowledge representation forms that need to be evaluated before a final selection of the storage-structure can be made. But, by resolving the storage-structure issue for the frame first, a large majority of knowledge representations used in existing expert systems may be able to be consolidated into one standard database.

While this recommendation involved other representation-forms, another recommendation that can be made involves the rule and network language-hierarchies (see Section IV). The recommendation is to provide a special-case definition for all of the characteristics of rules and networks given in Section IV. Then, using the same techniques given in Sections V-VIII, complete the formal analysis for these special language-hierarchies. If these special-case definitions could be made so that differences could be traced (e.g. the number of nested statements is "N"), then more than one special-case definition will

actually have been accounted for by this approach. However, these special-case definitions must be developed so that the language-type of the specially defined knowledge-language does not exceed the expected type of a knowledge-language generated from a more general definition. Otherwise, the results will not be typical of the more complex knowledge representations.

Additionally, it is important to notice that the open language-typing problem applies to both the knowledge-languages as well as the overall representation-languages. The specialized definition can only eliminate the existence portion of the problem. Finding the true language description still remains a risk due to Godel's theorem. However, if a language version could be found for this special case, then the conjectures of Section VIII would have additional concrete basis. This would add credibility to the overall representation findings of this dissertation without waiting for the knowledge representation researchers to reach a consensus on the form of knowledge.

Mentioning knowledge representation researchers brings to mind several recommendations for them specifically. One recommendation for these researchers is to provide detailed supporting proofs when they state that their knowledge representation is equivalent to others. While the findings of this dissertation showed that special equivalence cases exist (see Section VII), the findings also showed that complete equivalence of rules and networks does not exist

(see Proposition 8.06 of Section VIII). By showing that a production-rule language maps into a semantic-network language does not prove equivalence of the entire set of languages. The fact that a reverse map can be used to retrieve the original rules is not sufficient either. These researchers need to expand their representation-definitions in the manner used in this dissertation (see Section IV) to generate the general class definition for each representation-form. Then, they can use formal proof techniques similar to those of this dissertation (see Section VII) on these general classes to truly verify that class equivalence exists.

Another recommendation for these knowledge representation researchers is that they prove the correctness of the transformations they use when transforming from one representation to another. The finding of this dissertation that no single transformation exists for networks-to-rules supports the need for this added task (see Proposition 8.08 of Section VIII). The methods used in this dissertation (see Section VII) can be applied to help prove the correctness of specific transformations. However, these special transformations must not be used to transform a more general version of a knowledge representation without a similar generalization being applied in the correctness proofs of these special transformations. That is, one transformation may be proven correct for a representation containing a specific subset of knowledge-words, yet all of the

underlying characteristics of the overall knowledge-language may not have been taken directly into account in the original proof of correctness. Without expanding the proof of the special transformation to include all the words in the knowledge-language, representation-words that contain other knowledge-words could transform incorrectly when using this same transformation.

For example, let the general knowledge-language constitute a particular finite regular set. Now, let the specific word-set used to prove the correctness of a given network-to-rule transformation be a special regular subset of this general knowledge-language. Since both are regular sets over the same alphabet, a common mistake is to assume the same transformation would apply. However, as was shown in Section VII, some finite semantic-network languages exist that can never be transformed into a production-rule representation. Although, a semantic network formed from a subset of the knowledge-language can be transformed correctly. So, using the special transformation on the larger semantic-network representation would result in an incorrect production-rule representation being generated.

A further recommendation is that knowledge representation researchers get with the manufacturers of support tools and provide users the formal language description of the representation(s) that the tool will handle. As indicated in Section I, expert systems have to be validated for operational use. Since the knowledge representation used in

the knowledge-base (see Figure 4) has to be validated during the expert system's validation, users are faced with developing validation tests for their particular implementation of the representation. By using the formal language description along with their own implementation details, existing formal language algorithms (e.g. word membership) can be modified and used to perform validation tests. This can reduce the time needed to develop these tests.

### Closing

While the previous recommendations dealt with ways to apply, expand upon and improve upon the results of this dissertation, there is one final suggestion for future investigations in applying formal language theory to knowledge representations. This suggestion is to expand the analysis techniques to include computational time and space complexities. Formal language theory provides some of the guidelines that can be followed (see the references in Appendix A). Reference 105 provides other guidance for any special circumstances that might surface as a result of the detailed knowledge-word definitions.

This complexity information would be useful for those who develop support tools for these representations. Recalling that while this dissertation showed that no single support or automatic programming tool existed that could handle all knowledge-languages, there are tools that exist that support a countably infinite number of specific

implementation-versions for a given knowledge-language. Therefore, establishing complexity guidelines for these manufacturers to follow would result in a standard form of comparison among these different systems. That is, there may be several existing tools that could suffice for a particular application. However, one tool may be more efficient in both time and space than any of the others. This type of information would be invaluable in helping a user make a final tool selection.

With this last suggestion, the efforts of this dissertation draw to a close. The original goal of providing formality to the field of knowledge representation has been met. Even though open problems exist, it is important not to falsely conclude that formal language theory should not have been used to study knowledge representations. Godel's theorem does not say that every truth is unprovable. By using formal language theory, this dissertation has enhanced the knowledge about the various representation-forms. The formality provided by language theory allows objectivity to be used when reaching conclusions about knowledge representations. This is needed in light of the many subjective opinions of knowledge representation researchers (84).

While this dissertation has shown how to add formality to the knowledge representation field, it also has provided the foundation for an approach to study general computer data structures. Knowledge representations are just forms of abstract data types (3). By applying the same formal

language theory techniques to general abstract data types, it should be possible to gain formal knowledge about a wider variety of computer storage approaches than just representation-structures. Even though the dissertation does not provide absolute proof of the applicability of the approach to all cases, the positive results of this dissertation provide support for this conjecture.

While this dissertation has contributed to the formality of both the knowledge representation and the abstract data type areas of computer science, it has also contributed to the field of formal language theory itself. The dissertation provided the proof of the size of the formal language (and grammar, see Appendix D) set. Because of the insight gained from this proof, the complexity of the proofs of new formal language theorems can be reduced. Additionally, the dissertation expanded the centermarker-languages of Haines (98) to include any regular set as the centermarker. Finally, the dissertation provided examples of formal languages that actually exist outside of the theoretical environment. As a result, this dissertation provides support to the position that research on formal languages is more than just a theoretical curiosity.

QUOD ERAT DEMONSTRANDUM



## Appendix A: References on Formal Language Theory

### Languages and Grammars

- Aggarwal, Sarwan K. and James A. Heinen. "A General Class of Noncontext Free Grammars Generating Context Free Languages," Information and Control, 43: 187-194 (November 1979).
- Aho, Alfred V. "Nested Stack Automata," Journal of the ACM, 16: 383-406 (July 1969).
- Amar, V. and G. Putzolu. "On a Family of Linear Grammars," Information and Control, 7: 283-291 (September 1964).
- Araki, Toshiro and Nobuki Tokura. "Flow Language Equals Recursively Enumerable Languages." Acta Informatica, 15: 209-217 (June 1981).
- Autebert, Jean-Michel. "Pushdown-Automata and Families of Languages Generating Cylinders," Lecture Notes in Computer Science, Number 53, edited by J. Gruska. Berlin: Springer-Verlag, 1977.
- Baker, Brenda S. "Tree Transducers and Tree Languages," Information and Control, 37: 241-266 (June 1978).
- Baker, John L. "Grammars with Structured Vocabulary: A Model for the ALGOL-68 Definition," Information and Control, 20: 351-395 (May 1972).
- Bertoni, A. and others. "An Application of the Theory of Free Partially Commutative Monoids: Asymptotic Densities and Trace Languages," Lecture Notes in Computer Science, Number 118, edited by J. Gruska and M. Chytil. Berlin: Springer-Verlag, 1981.
- Boasson, L. "Classification of the Context-Free Languages." Lecture Notes in Computer Science, Number 53, edited by J. Gruska. Berlin: Springer-Verlag, 1977.
- Boasson, Luc and Maurice Nivat. "Adherences of Languages," Journal of Computer and System Sciences, 20: 285-309 (June 1980).
- Book, Ronald V. and Maurice Nivat. "Linear Languages and the Intersection Closures of Classes of Languages," SIAM Journal on Computing, 7: 167-177 (May 1978).

- Book, Ronald V. and Sheila A. Greibach. "Quasi-Realtime Languages," Mathematical Systems Theory, 4: 97-111 (1970).
- Chomsky, Noam. "A Note on Phrase Structure Grammars," Information and Control, 2: 393-395 (December 1959).
- . "Finite State Language," Information and Control 1: 91-112 (May 1958).
- Cohen, Rina S. and Arie Y. Gold. "Theory of  $\omega$ -Languages. II: A Study of Various Models of  $\omega$ -Type Generation and Recognition," Journal of Computer and System Sciences, 15: 185-208 (October 1977).
- . "Theory of  $\omega$ -Languages. I: Characterizations of  $\omega$ -Context-Free Languages," Journal of Computer and System Sciences, 15: 169-184 (October 1977).
- Crespi-Reggizzi and others. "Noncounting Context-Free Languages," Journal of the ACM, 25: 571-580 (October 1978).
- Culik, Karel, II and Rina Cohen. "LR-Regular Grammars--An Extension of LR(k) Grammars," Journal of Computer and System Sciences, 7: 66-96 (February 1973).
- Doberkat, Ernst-Erich. Lecture Notes in Computer Science, Number 113. Berlin: Springer-Verlag, 1981.
- Duske, J. and others. "Szilard Languages of IO-Grammars," Information and Control, 40: 319-331 (March 1979).
- Engelfriet, J. and G. Rozenberg. "Fixedpoint Languages, Equality Languages and Representation of Recursively Enumerable Languages," Journal of the ACM, 27: 499-518 (July 1980).
- . "Equality Languages and Fixed Point Languages," Information and Control, 43: 20-49 (October 1979).
- Fleck, A.C. "An Analysis of Grammars by Their Derivation Sets," Information and Control, 24: 389-398 (April 1974).
- Ginsburg, Seymour and Barbara Partee. "A Mathematical Model of Transformational Grammars," Information and Control, 15: 297-334 (October 1969).
- Giuliano, Joseph Alphonso. "Writing Stack Acceptors," Journal of Computer and System Sciences, 6: 168-204 (April 1972).

- Goldstine, Jonathan. "Some Independent Families of One-Letter Languages," Journal of Computer and System Sciences, 10: 351-369 (June 1975).
- Greibach, S.A. "Erasable Context-Free Languages," Information and Control, 29: 301-326 (December 1975).
- Gruska, J. "Some Classifications of Context-Free Languages," Information and Control, 14: 152-179 (February 1969).
- Harrison, Michael A. and Ivan M. Havel. "Real-Time Strict Deterministic Languages." SIAM Journal on Computing, 1: 333-349 (December 1972).
- Harrison, Michael A. and Mario Schkolnick. "A Grammatical Characterization of One-Way Nondeterministic Stack Languages," Journal of the ACM, 18: 148-172 (April 1971).
- Hart, Johnson M. "Acceptors for the Derivation Languages of Phase-Structure Grammars," Information and Control, 25: 75-92 (May 1974).
- Huang, T. and K.S. Fu. "On Stochastic Context-Free Languages," Information Sciences, 3: 201-224 (1971).
- Haussler, David. "Insertion Languages," Information Sciences, 31: 77-89 (October 1983).
- Haussler, David and H. Paulzeiger. "Very Special Languages and Representations of Recursively Enumerable Languages via Computation Histories," Information and Control, 47: 201-212 (December 1980).
- Hibbard, Thomas N. "Context-Limited Grammars," Journal of the ACM, 21: 446-453 (July 1974).
- Istrail, Sorin. "Elementary Bounded Languages," Information and Control, 39: 177-191 (November 1978).
- Joshi, A.K. and others. "String Adjunct Grammars: II. Equational Representation, Null Symbols, and Linguistic Relevance," Information and Control, 21: 235-260 (October 1972).
- "String Adjunct Grammars: I. Local and Distributed Adjunction," Information and Control, 21: 93-116 (September 1972).
- Joshi, Arvind K. and others. "Tree Adjunct Grammars," Journal of Computer and System Sciences, 10: 136-163 (February 1975).

- Kleijn, H.C.M. and G. Rozenberg. "Multi Grammars," International Journal of Computer Mathematics, 12: 177-201 (February 1983).
- "Sequential, Continuous and Parallel Grammars," Information and Control, 48: 221-260 (March 1981).
- Knast, Robert. "Finite-State Probabilistic Languages," Information and Control, 21: 148-170 (September 1972).
- Kobayashi, K. "Classification of Formal Languages by Functional Binary Transductions," Information and Control, 15: 95-109 (July 1969).
- Latteux, M. and G. Thierrin. "Semi-Discrete Context-Free Languages," International Journal of Computer Mathematics, 14: 3-18 (September 1983).
- Lee, E.T. and L.A. Zadeh. "Note on Fuzzy Languages," Information Sciences, 1: 421-434 (1969).
- Levy, Leon S. "Local Adjunct Languages and Regular Sets." SIAM Journal on Computing, 5: 305-308 (June 1976).
- Linna, M. "On  $\omega$ -Sets Associated with Context-Free Language," Information and Control, 31: 272-293 (July 1976).
- Matthews, G.H. "Two-Way Languages." Information and Control, 10: 111-119 (February 1967).
- Maurer, H.A. and others. "Finitary and Infinitary Interpretations of Languages," Mathematical Systems Theory, 15: 251-265 (July 1982).
- "Pure Grammars," Information and Control, 44: 47-72 (January 1980).
- Mazurkiewicz, Antoni W. "A Note on Enumerable Grammars," Information and Control, 14: 555-558 (June 1969).
- Meznik, Ivan. "G-Machine and Generable Sets," Information and Control, 20: 499-509 (June 1972).
- Mizumoto, M. and others. "B-Fuzzy Grammars," International Journal of Computer Mathematics, 4: 343-368 (December 1974).
- Poigne, Axel. "Context-Free Language on Infinite Words as Least Fixpoints." Lecture Notes in Computer Science, Number 117, edited by Ferenc Gecseg. Berlin: Springer-Verlag, 1981.

- Rabinovitz, Moshe. "Closure Properties and Languages Defined by Bilinear Automata," Journal of Computer and System Sciences, 17: 414-423 (December 1978).
- Revesz, Gyorgy. "Unilateral Context Sensitive Grammars and Left-to-Right Parsing," Journal of Computer and System Sciences, 5: 337-352 (August 1971).
- Ritchie, R.W. and F.N. Springsteel. "Language Recognition by Marking Automata," Information and Control, 20: 313-330 (May 1972).
- Rosenkrantz, D.J. and R.E. Stearns. "Properties of Deterministic Top-Down Grammars," Information and Control, 17: 226-256 (October 1970).
- Santos, Eugene S. "Fuzzy Automata and Languages," Information Sciences, 10: 195-197 (1977).
- "Max-Product Grammars and Languages," Information Sciences, 9: 1-23 (1975).
- "Realization of Fuzzy Languages by Probabilistic, Max-Product, and Maximum Automata," Information Sciences, 8: 39-53 (January 1975).
- "Context-Free Fuzzy Languages," Information and Control, 26: 1-11 (September 1974).
- "Regular Probabilistic Languages," Information and Control, 23: 58-70 (August 1973).
- "Probabilistic Grammars and Automata," Information and Control, 21: 27-47 (August 1972).
- Schlichtiger, Peter. "Partitioned Chained Grammars," Lecture Notes in Computer Science, Number 85, edited by J.W. de Bakker and J. van Leeuwen. Berlin: Springer-Verlag, 1980.
- Shyr, H.J. "Ordered Catenation and Regular Free Disjunctive Languages," Information and Control, 46: 257-269 (September 1980).
- Shyr, H.J. and G. Thierrin. "Power-Separating Regular Languages," Mathematical Systems Theory, 8: 90-95 (1974).
- Siromoney, Rani and Kamala Krithivasan. "Parallel Context-Free Languages," Information and Control, 24: 155-162 (February 1974).

- Skyum, Sven. "On Extensions of ALGOL-Like Languages," Information and Control, 26: 82-97 (September 1974).
- Staiger, Ludwig. "Finite-State Languages," Journal of Computer and System Sciences, 27: 434-448 (December 1983).
- Stearns, R.E. and P.M. Lewis. "Property Grammars and Table Machines," Information and Control, 14: 524-549 (June 1969).
- Takahashi, Masako. "Generalizations of Regular Sets and Their Application to a Study of Context-Free Languages," Information and Control, 27: 1-36 (January 1975).
- von Soloms, S.H. "The Characterization by Automata of Certain Classes of Languages in the Context Sensitive Area," Information and Control, 27: 262-217 (March 1975).
- Wagner, Klaus. "On  $\omega$ -Regular Sets," Information and Control, 43: 123-177 (November 1979).
- Walters, Daniel A. "Deterministic Context-Sensitive Languages: Part I," Information and Control, 17: 14-40 (August 1970).
- "Deterministic Context-Sensitive Languages: Part II," Information and Control, 17: 41-61 (August 1970).
- Williams, John H. "Bounded Context Parsable Grammars," Information and Control, 28: 314-334 (August 1975).
- Wise, David S. "Generalized Overlap Resolvable Grammars and Their Parsers," Journal of Computer and System Sciences, 6: 538-572 (December 1972).
- Workman, David A. "SR(s,k) Parsers: A Class of Shift-Reduce Bounded-Context Parsers," Journal of Computer and System Sciences, 22: 178-197 (April 1981).
- "Turn-Bounded Grammars and Their Relation to Ultralinear Languages," Information and Control, 32: 188-200 (October 1976).
- Wotschke, Detlef. "Degree-Languages: A New Concept of Acceptance," Journal of Computer and System Sciences, 14: 187-209 (April 1977).

## Inference

Angluin, Dana. "Inference of Reversible Languages," Journal of the ACM, 29: 741-765 (July 1982).

------. "Finding Patterns Common to a Set of Strings," Journal of Computer and System Sciences, 21: 46-62 (August 1980).

------. "Inductive Inference of Formal Languages from Positive Data," Information and Control, 45: 117-135 (May 1980).

Berger, J. and C. Pair. "Inference for Regular Bilanguages," Journal of Computer and System Sciences, 16: 100-122 (February 1978).

Case, John and Christopher Lynes. "Machine Inductive Inference and Language Identification," Lecture Notes in Computer Science, Number 140, edited by M. Nielsen and E.M. Schmidt. Berlin: Springer-Verlag, 1982.

Cook, Craig M. and others. "Grammatical Inferences by Hill Climbing," Information Sciences, 10: 59-80 (1976).

Feldman, Jerome. "Some Decidability Results on Grammatical Inference and Complexity," Information and Control, 20: 244-262 (April 1972).

Feliciangeli, Horacio and Gabor T. Herman. "Algorithms for Producing Grammars from Sample Derivation: A Common Problem for Formal Language Theory and Developmental Biology," Journal of Computer and System Sciences, 7: 97-118 (February 1973).

Floyd, Robert W. "A Note on Mathematical Induction on Phrase Structure Grammars," Information and Control, 4: 353-358 (December 1961).

Gold, E. Mark. "Language Identification in the Limit," Information and Control, 10: 447-474 (May 1967).

Heger, Josef Peter and others. "Inference of Deterministic One-Counter Languages," Information Sciences, 32: 139-163 (May 1984).

Knobe, Bruce and Kathleen Knobe. "A Method for Inferring Context-Free Grammars," Information and Control, 31: 129-146 (June 1976).

- Lu, Hao-Ru and King-Sun Fu. "A Generalized Approach to Inference of Context Free Programmed Grammars," IEEE Transactions on Systems, Man and Cybernetics, SMC-14: 191-202 (March/April 1984).
- Miclet, Laurent. "Regular Inference with a Tail Clustering Method," IEEE Transactions on Systems, Man and Cybernetics, SMC-10: 737-743 (November 1980).
- Osherson, Daniel N. and Scott Weinstein. "Criteria of Language Learning," Information and Control, 52: 123-138 (February 1982).
- Schuler, P.F. "Inductive Definability in Formal Language Theory," Journal of Computer and System Sciences, 16: 400-412 (June 1978).
- Shamir, E. "A Remark on Discovery Algorithms for Grammars," Information and Control, 5: 246-251 (September 1962).
- Wharton, R.M. "Grammar Enumeration and Inference," Information and Control, 33: 250-253 (March 1977).
- . "Approximate Language Identification," Information and Control, 26: 236-255 (November 1974).
- Wiehagen, Rolf. "Indentificaton of Formal Languages," Lecture Notes in Computer Science, Number 53, edited by J. Gruska. Berlin: Springer-Verlag, 1977.

#### Language Problems

- Albert, Jurgen and Derick Wood. "Checking Sets, Test Sets, Rich Languages and Commutatively Closed Languages," Journal of Computer and System Sciences, 26: 82-91 (February 1983).
- Angluin, Dana. "On The Complexity of Minimum Inference of Regular Sets," Information and Control, 39: 337-350 (December 1978).
- Asveld, Peter R.J. "Time and Space Complexities of Inside-Out Macro Languages," International Journal of Computer Mathematics, 10: 3-14 (October 1981).
- . "Space-Bounded Complexity Class and Iterated Deterministic Substitution," Information and Control, 44: 282-299 (March 1980).



- Baker, Brenda S. "Non-Context-Free Grammars Generating Context-Free Languages," Information and Control, 24: 231-246 (March 1974).
- Benson, David B. "Some Preservation Properties of Normal Form Grammars," SIAM Journal on Computing, 6: 381-402 (June 1977).
- Bertsch, Eberhard. "An Observation on Relative Parsing Time," Journal of the ACM, 22: 493-498 (October 1975).
- Blattner, Meera. "Structural Similarity in Context-Free Languages," Information and Control, 30: 267-294 (March 1976).
- Boasson, Luc and others. "The Rational Index: A Complexity Measure for Languages," SIAM Journal on Computing, 10: 284-296 (May 1981).
- Book, Ronald V. "On the Complexity of Formal Grammars," Acta Informatica, 9: 171-181 (1978).
- "On Languages Accepted in Polynomial Time" SIAM Journal on Computing, 1: 281-287 (December 1972).
- "Terminal Context in Context-Sensitive Grammars," SIAM Journal on Computing, 1: 20-30 (March 1972).
- Book, Ronald V. and Franz-Josef Brandenburg. "Equality Sets and Complexity Classes," SIAM Journal on Computing, 9: 729-743 (November 1980).
- Book, Ronald and others. "Reversal-Bounded Acceptors and Intersection of Linear Languages," SIAM Journal on Computing, 3:: 284-295 (December 1974).
- Brainerd, Walter S. and Ronald B. Knode. "Some Criteria for Determining Recognizability of a Set," Information and Control, 21: 171-184 (September 1972).
- Brandenburg, Franz J. "The Computational Complexity of Certain Graph Grammars," Lecture Notes in Computer Science, Number 145, edited by A.B. Cremers and H.P. Kriegel. Berlin: Springer-Verlag, 1983.
- Burkhard, W.A. and P.P. Varaiya. "Complexity Problems in Real Time Languages." Information Sciences, 3: 87-100 (1971).

- Buttelmann, H. William. "On the Syntactic Structure of Unrestricted Grammars. I. Generative Grammars and Phrase Structure Grammars," Information and Control, 29: 29-80 (September 1975).
- . "On the Syntactic Structure of Unrestricted Grammars. II. Automata," Information and Control, 29: 81-101 (September 1975).
- Cannon, Robert L. Jr. "Phrase Structure Grammars Generating Context-Free Languages," Information and Control, 29: 252-267 (November 1975).
- Chen, Kuo An and Ming-Kuei Hu. "A Finite State Probabilistic Automaton that Accepts a Context Sensitive Language that is not Context Free," Information and Control, 35: 196-208 (November 1977).
- Cohen, Joel M. "The Equivalence of two Concepts of Categorical Grammar", Information and Control, 10: 475-484 (May 1967).
- Cohen, Rina S. and Arie Y. Gold. " -Computations on Deterministic Pushdown Machines," Journal of Computer and System Sciences, 16: 275-300 (June 1978).
- Conner, William M. "The Dimension of a Formal Language," Information and Control, 29: 1-10 (September 1975).
- Cremers, A.B. and others. "On the Complexity of Regulated Context-Free Rewriting," Information and Control, 25: 10-19 (May 1974).
- Crespi-Reghizzi, Stefano and others. "Algebraic Properties of Operator Precedence Languages," Information and Control, 37: 115-133 (May 1978).
- Culik, Karel, II. and Tero Harju. "The -Sequence Equivalence Problem for DOL Systems is Decidable," Journal of the ACM, 31: 282-298 (April 1984).
- Duris, Pavol and Zvi Galil. "A Time-Space Tradeoff for Language Recognition," Mathematical Systems Theory, 17: 3-12 (April 1984).
- . "On Reversal-Bounded Counter Machines and on Pushdown Automata with a Bound on the Size of the Pushdown Store," Information and Control, 54: 217-227 (September 1982).

- Ellis, Clarence A. "The Halting Problem for Probabilistic Context-Free Generators," Journal of the ACM, 19: 396-399 (July 1972).
- Engelfriet, Joost and Gilberto File. "Passes and Paths of Attribute Grammars," Information and Control, 49: 125-169 (May 1981).
- Gallaire, Herve. "Recognition Time on Context-Free Languages by On-Line Turing Machines," Information and Control, 15: 288-295 (September 1969).
- Georgeff, M.P. "Interdependent Translation Schemes," Journal of Computer and System Sciences, 22: 198-219 (April 1981).
- Ginsburg, Seymour and G.F. Rose. "Operations Which Preserve Definability in Languages." Journal of the ACM, 10: 175-195 (April 1963).
- Ginsburg, Seymour and Michael Harrison. "On the Closure of AFL under Reversal," Information and Control, 17: 395-409 (November 1970).
- Ginsburg, Seymour and Nancy Lynch. "Derivation Complexity in Context-Free Grammar Forms," SIAM Journal on Computing, 6: 123-138 (March 1977).
- Ginsburg, Seymour and Sheila Greibach. "On AFL Generators for Finitely Encoded AFA," Journal of Computer and System Sciences, 7: 1-27 (February 1973).
- "Mappings which Preserve Context Sensitive Languages," Information and Control, 9: 563-582 (December 1966).
- Ginsburg, Seymour and others. "On the Equality of Grammatical Families," Journal of Computer and System Sciences, 26: 171-196 (April 1983).
- Graham, Susan L. "On Bounded Right Context Languages and Grammars," SIAM Journal on Computing, 3: 224-254 (September 1974).
- Greibach, Sheila A. "One Counter Languages and the IRS Condition," Journal of Computer and System Sciences, 10: 237-247 (April 1975).
- "The Hardest Context-Free Language," SIAM Journal on Computing, 2: 304-310 (December 1973).

- Greibach, Sheila A. "The Undecidability of the Ambiguity Problem for Minimal Linear Grammars," Information and Control, 6: 119-125 (June 1963).
- Greibach, S.A. and E.P. Friedman. "Super Deterministic PDAs: A Subcase with a Decidable Inclusion Problem," Journal of the ACM, 27: 675-700 (October 1980).
- Hashiguchi, K. "Representation Theorem on Regular Languages," Journal of Computer and System Sciences, 27: 101-115 (August 1983).
- Hart, Johnson M. "Derivation Structures for Strictly Context-Sensitive Grammars," Information and Control, 45: 68-89 (April 1980).
- Hartmanis, J. "On the Succinctness of Different Representations of Languages," SIAM Journal on Computing, 9: 114-120 (February 1980).
- Hartmanis, J. and J.E. Hopcroft. "What Makes Some Language Theory Problems Undecidable," Journal of Computer and System Sciences, 4: 368-376 (August 1970).
- Huang, T. and K.S. Fu. "On Stochastic Context-Free Languages," Information Sciences 3: 201-224 (1971).
- Hughes, Charles E. "Derivatives and Quotients of Prefix-Free Context-Free Languages," Information and Control, 45: 229-235 (June 1980).
- Hunt, H.B., III. "On the Decidability of Grammar Problems," Journal of the ACM, 29: 429-447 (April 1982).
- Hunt, Harry B., III and others. "On the Equivalence, Containment, and Covering Problems for the Regular and Context-Free Languages," Journal of Computer and System Sciences, 12: 222-268 (April 1976).
- Hutchins, Sandra E. "Moments of String and Derivation Lengths of Stochastic Context-Free Grammars," Information Sciences, 4: 179-191 (1972).
- Huynh, Dung T. "Commutative Grammars: The Complexity of Uniform Word Problems," Information and Control, 57: 21-39 (April 1983).
- Igarashi, Y. "Tape Bounds for Some Subclasses of Deterministic Context-Free Languages," Information and Control, 37: 321-333 (June 1978).

- Janssens, D. and G. Rozenberg. "Decision Problems for Node Label Controlled Graph Grammars," Journal of Computer and System Sciences, 22: 144-177 (April 1981).
- Jones, Neil D. "A Note on the Index of a Context-Free Language," Information and Control, 16: 201-202 (April 1970).
- ". "Context-Free Languages and Rudimentary Attributes," Mathematical Systems Theory, 3:: 102-108 (1969).
- Kamimura, Tsutomu and Giora Slutzki. "Transduction of Dags and Trees," Mathematical Systems Theory, 15: 225-249 (July 1982).
- Kaminger, F.P. "The Noncomputability of the Channel Capacity of Context-Sensitive Languages," Information and Control, 17: 175-182 (September 1970).
- Kuich, Werner. "The Complexity of Skewlinear Tuple Languages and o-Regular Languages," Information and Control, 19: 353-367 (November 1971).
- ". "On the Entropy of Context-Free Languages," Information and Control, 16: 173-200 (April 1970).
- Kuich, Werner and Hermann Maurer. "The Structure Generating Function and Entropy of Tuple Languages," Information and Control, 19: 195-203 (October 1971).
- Kuich, Werner and R.K. Shyamasundar. "The Structure Generating Function of Some Families of Languages," Information and Control, 32: 85-92 (September 1976).
- Kobayashi, K. "Structural Complexity of Context-Free Language," Information and Control, 18: 299-310 (May 1971).
- Kobuchi, Y. and D. Wood. "On the Complete Simulation of DOL Scheme and Locally Catenative Schemes," Information and Control, 49: 64-80 (April 1981).
- Kortelainen, Juha. "Some Properties of Language Families Generated by Commutative Languages," Lecture Notes in Computer Science, Number 117, edited by Ferenc Gecseg. Berlin: Springer-Verlag, 1981.
- Kosaraju, S. Rao. "Probabilistic Automata--A Problem of Paz," Information and Control, 23: 97-104 (August 1973).

- Kuroda, S.-Y. "A Topological Study of Phrase-Structured Languages," Information and Control, 30: 307-379 (April 1976).
- Latteux, M. and G. Rozenberg. "Commutative One-Counter Languages are Regular." Journal of Computer and System Sciences, 29: 54-57 (August 1984).
- Levis, F.D. "A Note on Context Free Languages, Complexity Classes, and Diagonalization," Mathematical Systems Theory, 14: 223-227 (July 1981).
- Levy, Leon S. "Structural Aspects of Local Adjunct Language," Information and Control, 23: 260-287 (October 1973).
- Linna, Matti. "On the Regularity Problem of SF-Languages Generated by Minimal Linear Grammars," Lecture Notes in Computer Science, Number 117, edited by Ferenc Gecseg. Berlin: Springer-Verlag, 1981.
- Maggiulo-Schettim, Andrea and Jozef Winkowski. "Processes of Transforming Structures," Journal of Computer and System Sciences, 24: 245-282 (June 1982).
- Maibaum, T.S.E. "Pumping Lemmas for Term Languages," Journal of Computer and System Sciences, 17: 319-330 (December 1978).
- Maurer, H.A. and others. "Completeness of Context-Free Grammar Forms," Journal of Computer and System Sciences, 23: 1-10 (August 1981).
- Moriy, Etsuro. "Associate Languages and Derivational Complexity of Formal Grammars and Languages," Information and Control, 22: 139-162 (March 1973).
- Nasu, Masakazu and Namio Honda. "A Context-Free Language which is not Acceptable by a Probabilistic Automaton," Information and Control, 18: 233-236 (April 1971).
- Parikh, Rohit J. "On Context-Free Languages," Journal of the ACM, 13: 570-581 (October 1966).
- Paull, Marvin C. and Stephen H. Unger. "Structural Equivalence of Context-Free Grammars," Journal of Computer and System Sciences, 2: 427-463 (December 1968).
- Paun, Gheorghe. "Six Nonterminals are Enough for Generating each R.E. Language by a Matrix Grammar," International Journal of Computer Mathematics, 15: 23-37 (April 1984).

- Paun, Gheorghe. "On the Generative Capacity of Conditional Grammars," Information and Control, 43: 178-186 (November 1979).
- . "On the Index of Grammars and Languages," Information and Control, 35: 259-266 (December 1977).
- . "The Generative Capacity of the Compound Grammars," Information and Control, 34: 50-54 (May 1977).
- Rozenberg, G. and D. Vermeir. "On the Effect of the Finite Index Restriction on Several Families of Grammars," Information and Control, 39: 284-302 (December 1978).
- Salomaa, Arto. "The Generative Capacity of Transformational Grammars of Ginsburg and Partee," Information and Control, 18: 227-232 (April 1971).
- Santos, Eugene S. "A Note on Probabilistic Grammars," Information and Control, 25: 393-394 (August 1974).
- . "A Note on Bracketed Grammars," Journal of the ACM, 19: 222-224 (April 1972).
- Savitch, Walter J. "How to Make Arbitrary Grammars Look Like Context-Free Grammars," SIAM Journal on Computing, 2: 174-182 (September 1973).
- Soule, Stephen. "Entropies of Probabilistic Grammars," Information and Control, 25: 57-74 (May 1974).
- Subramanian, K.G. and Rani Siromoney. "On the Generative Capacity of Compound String and Array Grammars," Information Sciences, 26: 231-241 (April 1982).
- Takaoka, Tadao. "A Definition of measures over Language Space." Journal of Computer and System Sciences, 17: 376-387 (December 1978).
- Taniguchi, Kenichi and Tadao Kasami. "Reduction of Context-Free Grammars," Information and Control, 17: 92-108 (August 1970).
- . "A Note on Computing Time for the Recognition of Context-Free Languages by a Single-Tape Turing Machine," Information and Control, 14: 278-284 (March 1969).
- Turakainen, Paavo. "On Some Bounded SemiAFLs and AFLs," Information Sciences, 23: 31-48 (February 1981).

Turakainen, Paavo. "On Homomorphic Images of Rational Stochastic Languages," Information and Control, 30: 96-105 (January 1976).

----- . "Some Closure Properties of the Family of Stochastic Languages," Information and Control, 18: 253-256 (April 1971).

----- . "On m-Adic Stochastic Languages," Information and Control, 17: 410-415 (November 1970).

Ukkonen, Esko. "Lower Bounds on the Size of Deterministic Parsers," Journal of Computer and System Sciences, 26: 153-170 (April 1983).

Valiant, Leslie G. "A Note on the Succinctness of Descriptions of Deterministic Languages," Information and Control, 32: 139-145 (October 1976).

Walter, Herman. "Topologies on Formal Languages," Mathematical Systems Theory, 9: 142-158 (1975).

Yntema, M.K. "Cap Expressions for Context-Free Languages," Information and Control, 18: 311-318 (May 1971).

#### Relationship to Other Areas

Claus, Volker and others, editors. Lecture Notes in Computer Science, Number 73. Berlin: Springer-Verlag, 1978.

Cook, Curtis R. "First Order Graph Grammars," SIAM Journal on Computing, 3: 90-99 (March 1974).

Crespi-Reghizzi, S. and D. Mandrioli. "Petri Nets and Szilard Languages," Information and Control, 33: 177-192 (February 1977).

Dassow, Jurgen. "A Note on Programmed OL Systems," Information Sciences, 30: 1-4 (July 1983).

Ehrenfeucht, A. and G. Rozenberg. "On Basic Properties of DOS Systems and Languages," Information and Control, 47: 137-153 (November 1980).

Ehrenfeucht, A. and others. "Context-Free Normal Systems and ETOL Systems," Journal of Computer and System Sciences, 26: 34-46 (February 1983).



- Ehrenfeucht, A. and others. "Continuous Grammars," Information and Control, 46: 71-91 (July 1980).
- Frougny, Christiane and others. "On The Holtz Group of a Context-Free Grammar," Acta Informatica, 18: 109-115 (November 1982).
- Herman, Gabor T. "A Biologically Motivated Extension of ALGOL-Like Languages," Information and Control, 22: 487-502 (June 1973).
- Janssens, D. and G. Rozenberg. "On the Structure of Node-Label-Controlled Graph Languages," Information Sciences, 20: 191-216 (April 1980).
- ". "Restrictions, Extensions, and Variations of NLC Grammars," Information Sciences, 20: 217-244 (April 1980).
- Kleijn, H.C.M. and G. Rozenberg. "A Study in Parallel Rewriting Systems," Information and Control, 44: 134-163 (February 1980).
- Kuich, Werner. "Generating Functions for Derivation Trees," Information and Control, 45: 199-138 (May 1980).
- Lee, K.P. and G. Rozenberg. "TIL System and Languages," Information Sciences, 12: 203-227 (1977).
- Maibaum, T.S.E. "A Generalized Approach to Formal Languages," Journal Computer and System Sciences, 8: 409-439 (June 1974).
- Maurer, H.A. and others. "On the Complexity of the General Coloring Problem," Information and Control, 51: 128-145 (November 1981).
- Moshell, J.M. and J. Rothstein. "Bus Automata and Immediate Languages," Information and Control, 40: 88-121 (January 1979).
- Nishio, Hidenosuke. "A Automaton Generating Series of Graphs," Information Sciences, 25: 153-174 (November 1981).
- Pratt, Terrence W. "Pair Grammars, Graph Languages and String-to-Graph Translations," Journal of Computer and System Sciences, 5: 560-595 (December 1971).

- Rajasethupathy, K.S. and R.K. Shyamasundar. "Programmed OL-Systems," Information Sciences, 20: 137-150 (March 1980).
- Rosen, Barry K. "Deriving Graphs from Graphs by Applying A Production," Acta Informatica, 4: 337-357 (1975).
- Rozenberg, Grzegorz. "TOL Systems and Languages," Information and Control, 23: 357-381 (April 1974).
- Rozenberg, G. and A.H. von Solms. "Rewriting Systems with a Clocking Mechanism," Information Sciences, 14: 221-280 (June 1978).
- Rozenberg, G. and A. Salomaa. "New Squeezing Mechanisms for L Systems," Information Sciences, 12: 187-201 (1977).
- Rozenberg, G. and D. Vermeir. "On ETOL System of Finite Index," Information and Control, 38: 103-133 (July 1978).
- Rozenberg, G. and K.P. Lee. "Some Properties of the Class of L Languages with Interactions," Journal of Computer and System Sciences, 11: 129-147 (August 1975).
- Rozenberg, G. and P.G. Doucet. "On OL-Languages," Information and Control, 19: 302-318 (November 1971).
- Rosenkrantz, Daniel J. "Matrix Equations and Normal Forms for Context-Free Grammars," Journal of the ACM, 14: 501-507 (July 1967).
- Salomon, Kenneth B. "String and Graph Grammar Characterization of Bounded Regular Languages," International Journal of Computer and Information Sciences, 7: 1-10 (March 1978).
- Sanders, Jerry. "Document Association and Classification Based on L-Languages," Journal of the ACM, 12: 249-253 (April 1965).
- Schneider, H.J. and H. Ehrig. "Grammars on Partial Graphs," Acta Informatica, 6: 297-316 (1976).
- Siromoney, Rani and Gift Siromoney. "Extended Controlled Table L-Arrays," Information and Control, 35: 119-138 (October 1977).
- Skyum, Sven. "Decomposition Theorems for Various Kinds of Languages Parallel in Nature," SIAM Journal on Computing, 5: 284-296 (June 1976).

Streinu, Ileana. "Grammar Directed Godel Numberings,"  
International Journal of Computer Mathematics, 14:  
223-237 (December 1983).

Urponen, Topi. "Equations with a Dyck Language Solution,"  
Information and Control, 30: 21-37 (January 1976).

Valk, Rudiger and Guy Vidal-Naquet. "Petri Nets and Regular  
Languages," Journal of Computer and System Sciences, 23:  
299-325 (December 1981).

Vannakakis, Mihalis and Theodosios Pavlidis. "Topological  
Characterization of Families of Graphs Generated by  
Certain Types of Graph Grammars," Information and  
Control, 42: 72-86 (July 1979).

Vigna, Pierluigi Della and Carlo Ghezzi. "Context-Free  
Graph Grammars," Information and Control, 37: 207-233  
(May 1978).

Vitanyi, Paul M.B. "Context Sensitive Table Lindenmayer  
Languages and a Relation to the LBA Problem,"  
Information and Control, 33: 217-226 (March 1977).

Vitanyi, Paul M.B. and Adrian Walker. "Stable String  
Languages of Lindenmayer Systems," Information and  
Control, 37: 134-149 (May 1978).

von Solms, S.H. "Node-Label Controlled Graph Grammars with  
Context Conditions," International Journal of Computer  
Mathematics, 15: 39-49 (April 1984).

Witt, Kurt-Ulrich. "Finite Graph-Acceptors and Regular  
Graph-Languages," Information and Control, 50: 242-258  
(September 1981).

Yokomori, Takashi. "Stochastic Characterizations of EOL  
Languages," Information and Control, 45: 26-33  
(April 1980).

#### Regulated Languages

Aho, A.V. and others. "Weak and Mixed Strategy Precedence  
Parsing," Journal of the ACM, 19: 225-243 (April 1972).

Barth, Gerhard. "Grammars with Dynamic Control Sets,"  
Lecture Notes in Computer Science, Number 62, edited by  
G. Ausiello and C. Bohm. Berlin: Springer-Verlag, 1978.

- Cremers, A.B. and O. Mayer. "On Vector Languages," Journal of Computer and System Sciences, 8: 158-166 (April 1974).
- . "On Matrix Languages," Information and Control, 23: 86-96 (August 1973).
- Crespi-Reghizzi, Stefano and others. "Operator Precedence Grammars and the Noncounting Property," SIAM Journal on Computing, 10: 174-191 (February 1981).
- Duske, J. and others. "IO-Macrolanguages and Attributed Translations," Information and Control, 35: 87-105 (October 1977).
- Engelfriet, Joost and Erik Meineche Schmidt. "IO and OI. II.," Journal of Computer and System Sciences, 16: 67-99 (February 1978).
- . "IO and OI. I.," Journal of Computer and System Sciences, 15: 328-353 (December 1977).
- Engelfriet, Joost and Gilbero File. "Simple Multi-Visit Attribute Grammars," Journal of Computer and System Sciences, 24: 283-314 (June 1982).
- Engelfriet, Joost and Giora Slutzki. "Extended Macro Grammars and Stack Controlled Machines," Journal of Computer and System Sciences, 29: 366-408 (December 1984).
- . "Bounded Nesting in Macro Grammars," Information and Control, 42: 157-193 (August 1979).
- Engelfriet, Joost and others. "Stack Machines and Classes of Nonnested Macro Language," Journal of the ACM, 27: 96-117 (January 1980).
- Fris, Ivan. "Grammars with Partial Ordering of the Rules," Information and Control, 12: 415-425 (May/June 1968).
- Gilbert, Philip. "On Syntax of Algorithmic Languages," Journal of the ACM, 13: 90-107 (January 1966).
- Ginsburg, Seymour and Edwin H. Spanier. "Control Sets on Grammars," Mathematical Systems Theory, 2: 159-177 (1968).
- Golan, Igal. "Conditional Grammars," Journal of Computer and System Sciences, 15: 354-371 (December 1977).

- Greibach, S.A. "Control Sets on Context-Free Grammar Forms," Journal of Computer and System Sciences, 15: 35-98 (August 1977).
- Guha, Ratan K. and Raymond T. Yeh. "Periodic Representation of Equal Matrix Grammars," Information and Control, 22: 435-446 (June 1973).
- Ibarra, Oscar H. "Simple Matrix Languages," Information and Control, 17: 359-394 (November 1970).
- Kasai, Takumi. "A Universal Context-Free Grammar," Information and Control, 28: 30-34 (May 1975).
- Knuth, Donald E. "Semantics of Context-Free Languages," Mathematical Systems Theory, 2: 127-145 (1968).
- Lepisto, Timo. "On Ordered Context-Free Grammars," Information and Control, 22: 56-68 (February 1973).
- Luker, M. "Control Sets on Grammar Using Depth-First Derivations," Mathematical Systems Theory, 13: 349-359 (1980).
- Maurer, H.A. "Simple Matrix Languages with a Left Most Restriction," Information and Control, 23: 128-139 (September 1973).
- Mayer, O. "Some Restrictive Devices for Context-Free Grammars," Information and Control, 20: 69-92 (February 1972).
- Moriya, Etsuro. "Some Remarks on State Grammars and Matrix Grammars," Information and Control, 23: 48-57 (August 1973).
- Parchmann, R. and others. "On Deterministic Indexed Languages," Information and Control, 45: 48-67 (April 1980).
- Pascu, Anca and Gheorghe Paun. "A Homomorphic Representation of Simple Matrix Languages," Information and Control, 35: 267-275 (December 1977).
- Paun, Gheorghe. "On the Family of Finite Index Matrix Languages," Journal of Computer and System Sciences, 18: 267-280 (June 1979).
- Rajlich, Vaclav. "Absolutely Parallel Grammars and Two-Way Finite-State Transducers," Journal of Computer and System Sciences, 6: 324-342 (August 1972).

Rozenberg, G. "A Note on Universal Grammars," Information and Control, 34: 172-175 (June 1977).

----- "Direction Controlled Programmed Grammars," Acta Informatica, 1: 242-252 (1972).

Rozenberg, G. and A. Salomaa. "Context-Free Grammar with Graph-Controlled Tables," Journal of Computer and System Sciences, 13: 98-99 (August 1976).

Saarinen, Mikko. "On Constructing Efficient Evaluators for Attribute Grammars," Lecture Notes in Computer Science, Number 62, edited by G. Ausiello and C. Bohm. Berlin: Springer-Verlag, 1978.

Salomaa, Arto. "Matrix Grammars with a Left Most Restriction," Information and Control, 20: 143-149 (March 1972).

----- "Periodically Time-Variant Context-Free Grammars," Information and Control, 17: 294-311 (October 1970).

----- "Probabilistic and Weighted Grammars," Information and Control, 15: 529-544 (December 1969).

----- "On the Index of a Context-Free Grammar and Language," Information and Control, 14: 474-477 (May 1969).

Siromoney, Rani. "Finite-Turn Checking Automata," Journal of Computer and System Sciences, 5: 549-559 (December 1971).

----- "On Equal Matrix Languages," Information and Control, 14: 135-151 (February 1969).

Watt, David Anthony. "The Parsing Problem for Affix Grammars," Acta Informatica, 8: 1-20 (1977).

Wood, Derick. "A Note on Bicoloured Digraph Grammar Systems," International Journal of Computer Mathematics, 3: 301-308 (December 1973).

Wotschke, Eva-Maria Muckstein and others. "Size, Index, and Context-Sensitivity of Controlled Partition Grammars," Mathematical Systems Theory, 11: 47-60 (1977).

#### Abstract Families of Languages

Boasson, L. "Classification of the Context-Free Languages," Lecture Notes in Computer Science, Number 53, edited by J. Gruska. Berlin: Springer-Verlag, 1977.

- Book, Ronald V. and Ben Wegbreit. "A Note on AFLs and Bounded Erasing," Information and Control, 19: 18-29 (August 1971).
- Ginsburg, Seymour and Edwin H. Spanier. "On Incomparable Abstract Family of Languages (AFL)," Journal of Computer and System Sciences, 9: 88-108 (August 1974).
- ". "AFL with the Semilinear Property," Journal of Computer and System Sciences, 5: 365-396 (August 1971).
- ". "Substitution in Families of Languages," Information Sciences, 2: 83-110 (1970).
- Ginsburg, Seymour and Gene F. Rose. "On the Existence of Generators for Certain AFLs," Information Sciences, 2: 431-446 (1970).
- Ginsburg, Seymour and Jonathan Goldstine.  
"Intersection-Closed Full AFL and the Recursively Enumerable Languages," Information and Control, 22: 201-231 (April 1973).
- Goldstine, Jonathan. "Bounded AFLs," Journal of Computer and System Sciences, 12: 399-419 (June 1976).
- ". "Substitution and Bounded Languages," Journal of Computer and System Sciences, 6: 9-29 (February 1972).
- Greibach, S.A. "Erasing in Context-Free AFLs," Information and Control, 21: 436-465 (December 1972).
- ". "Characteristics and Ultrarealtime Languages," Information and Control, 18: 65-98 (February 1971).
- ". "Checking Automata and One-Way Stack Languages," Journal of Computer and System Sciences, 3: 196-217 (May 1969).
- Greibach, Sheila and Seymour Ginsburg. "Multitape AFA," Journal of the ACM, 19: 193-221 (April 1972).
- Moriya, Etsuro. "Characterization Theorems on Abstract Families of Transducers," Information Sciences, 9: 227-238 (1975).
- Rovan, Branislav. "A Framework for Studying Grammars," Lecture Notes in Computer Science, Number 118, edited by J. Gruska and M. Chytil. Berlin: Springer-Verlag, 1981.

Tokomori, Takashi and Derick Wood. "An Inverse Homomorphic Characterization of Full Principle AFL," Information Sciences, 33: 209-216 (September 1984).

Turakainen, Paavo. "A Homomorphic Characterization of Principle SemiAFLs without Using Intersection with Regular Sets," Information Sciences, 27: 141-149 (July 1982).

### Hierarchical Languages

Brzozowski, J.A. and R. Knast. "The Dot-Depth Hierarchy of Star-Free Language is Infinite," Journal of Computer and System Sciences, 16: 37-55 (February 1978).

Cremers, A.B. "Nonsequential Languages," Information and Control, 23: 140-151 (September 1973).

Engelfriet, Joost. "Three Hierarchies of Transducers," Mathematical Systems Theory, 15: 95-125 (May 1982).

Ehrenfeucht, A. and others. "On ETOL Systems with Rank," Journal of Computer and System Sciences, 19: 237-255 (December 1979).

Fusaoka, Akira. "A Note on a Decomposition Theorem for Simple Deterministic Languages," Information and Control, 19: 272-274 (October 1971).

Ginsburg, Seymour and Edwin H. Spanier. "Derivation-Bounded Languages," Journal of Computer and System Sciences, 2: 228-250 (October 1968).

Ginsburg, Seymour and others. "A Prime Decomposition Theorem for Grammatical Families," Journal of Computer and System Sciences, 24: 315-361 (June 1982).

Greibach, S.A. "Jump PDA's and Hierarchies of Deterministic Context-Free Languages," SIAM Journal on Computing, 3: 111-127 (June 1974).

----- "Syntactic Operators on Full SemiAFLs," Journal of Computer and System Sciences, 6: 30-76 (February 1972).

----- "Chains of Full AFL's," Mathematical Systems Theory, 4: 231-242 (1970).

----- "An Infinite Hierarchy of Context-Free Languages," Journal for the ACM, 16: 91-106 (January 1969).



- Grosky, William I. "On the Power of Two-Dimensional Grammars which are not Length Consistant," Information and Control, 44: 197-221 (February 1980).
- Gruska, J. "A Few Remarks on the Index of Context-free Grammars and Languages," Information and Control, 19: 216-223 (October 1971).
- Harrison, Michael A. and Amiram Yehudai. "A Hierarchy of Deterministic Languages," Journal of Computer and System Sciences, 19: 63-78 (August 1979).
- Harrison, Michael A. and Ivan M. Havel. "Strict Deterministic Grammars," Journal of Computer and System Sciences, 7: 237-277 (June 1973).
- Hart, Johnson M. "An Infinite Hierarchy of Linear Local Adjunct Languages," Information and Control, 23: 245-259 (October 1973).
- Igarashi, Yoshihide and Namio Honda. "On the Extension of Gladkij's Theorem and the Hierarchies of Languages," Journal of Computer and System Sciences, 7: 199-217 (April 1973).
- Inoue, Katsushi and others. "A Space-Hierarchy Result on Two-Dimensional Alternating Turing Machines with Only Universal States," Information Sciences, 35: 79-90 (March 1985).
- Janiga, Ladislav. "Another Hierarchy Defined by Multiheaded Finite Automata," Lecture Notes in Computer Science, Number 118, edited by J. Gruska and M. Chytil. Berlin: Springer-Verlag, 1981.
- Khabbaz, Nabil A. "A Geometric Hierarchy of Languages," Journal of Computer and System Sciences, 8: 142-157 (April 1974).
- Liu, Leonard Y. and Peter Weiner. "An Infinite Hierarchy of Intersection of Context-Free Languages," Mathematical Systems Theory, 7: 185-192 (1973).
- Maurer, H.A. and D. Wood. "On Grammar Forms with Terminal Context," Acta Informatica, 6: 397-402 (1976).
- Maurer, H.A. and others. "Dense Hierarchies of Grammatical Families," Journal of the ACM, 29: 118-126 (January 1982).
- . "MSW Spaces," Information and Control, 46: 187-199 (September 1980).

Platek, Martin and Petr Sgall. "A Scale of Context Sensitive Languages: Applications to Natural Language," Information and Control, 38: 1-20 (July 1978).

Thomas, Wolfgang. "A Hierarchy of Sets of Infinite Trees," Lecture Notes in Computer Science, Number 145, edited by A.B. Cremers and H.P. Kriegel. Berlin: Springer-Verlag, 1983.

Vaishnau, Vijay Kumar and Sanat K. Basu. "On Coupled Languages and Translations," International Journal of Computer Mathematics, 6: 33-54 (March 1977).

Yehdai, Amiram. "A Hierarchy of Real-Time Deterministic Languages and Their Equivalence," Journal of Computer and System Sciences, 24: 91-100 (February 1982).

### Generalized Languages

Cremers, Armin and Seymour Ginsburg. "Context-Free Grammar Forms," Journal of Computer and System Sciences, 11: 86-117 (August 1975).

Ginsburg, Seymour and Edwin H. Spanier. "Substitution of Grammar Forms," Acta Informatica, 5: 377-386 (1975).

Kelemenova, Alica. "Grammatical Levels of the Position Restricted Grammars," Lecture Notes in Computer Science, Number 118, edited by J. Gruska and M. Chytil. Berlin: Springer-Verlag, 1981.

Maurer, H.A. and others. "Uniform Interpretation of Grammar Forms," SIAM Journal on Computing, 10: 483-502 (August 1981).

----- "Context-Free Grammar Forms with Strict Interpretations," Journal of Computer and System Sciences, 21: 110-135 (August 1980).

Mizumoto, Masaharu and others. "General Formulation of Formal Grammars," Information Sciences, 4: 87-100 (1972).

Vaishnavi, V.K. and D. Wood. "An Approach to a Unified Theory of Grammar and L Forms," Information Sciences, 15: 77-94 (July 1978).

Wood, Derick. Lecture Notes in Computer Science, Number 91. Berlin: Springer-Verlag, 1980.

## Appendix B: References on Knowledge Representation

### Predicate Calculus

- Blair, Howard A. "The Reasoning-Theoretic Complexity of the Semantics of Predicate Logic as a Programming Language," Information and Control, 54: 25-47 (July/August 1982).
- Dahl, Veronica. "Logic Programming as a Representation of Knowledge," Computer, 16: 106-111 (October 1983).
- Elcock, E.W. "How Complete Are Knowledge-Representation Systems?" Computer, 16: 114-118 (October 1983).
- Falaschi, M. and others. "A Synchronization Logic: Axiomatics and Formal Semantics of Generalized Horn Clauses," Information and Control, 60: 36-69 (January/February/March 1984).
- Israel, David J. "The Role of Logic in Knowledge Representation," Computer, 16: 37-41 (October 1983).
- . "What's Wrong with Non-monotonic Logic?" Proceedings of the First Annual National Conference on Artificial Intelligence, AAAI-80. 99-101. William Kaufmann, Inc., Los Altos CA, 1980.
- Levesque, Hector J. "The Interaction with Incomplete Knowledge Bases: A Formal Treatment," Proceedings of the Seventh International Joint Conference on Artificial Intelligence, IJCAI-81, Volume 1. 240-245. William Kaufmann, Inc., Los Altos CA, 1981.
- McDermott, Drew. "Nonmonotonic Logic II: Nonmonotonic Modal Theories," Journal of the ACM, 29: 33-57 (January 1982).
- McDermott, Drew and Jon Doyle. "Non-Monotonic Logic I," Artificial Intelligence, 13: 41-72 (April 1980).
- Nutter, Jane Terry. "Default Reasoning Using Monotonic Logic," Proceedings of the Third National Conference on Artificial Intelligence, AAAI-83. 297-300. William Kaufmann, Inc., Los Altos CA, 1983.
- Ohsuga, Setsuo. "Theoretical Basis for a Knowledge Representation System," Proceedings of the Sixth International Joint Conference on Artificial Intelligence, IJCAI-79, Volume 2. 676-683. William Kaufmann, Inc., Los Altos CA, 1979.

Sandewall, Erick. "Conversion of Predicate-Calculus Axioms, Viewed as Non-Deterministic Programs, to Corresponding Deterministic Programs," Advance Papers of the Third International Joint Conference on Artificial Intelligence, IJCAI-73. 230-234. William Kaufmann, Inc., Los Altos CA, 1973.

----- . "Representing Natural Language Information in Predicate Calculus," Machine Intelligence 6, edited by Bernard Meltzer and Donald Michie. New York: American Elsevier Publishing Company, Inc., 1971.

van Emden, M.H. "Programming with Resolution Logic," Machine Intelligence 8, edited by E.W. Elcock and Donald Michie. New York: John Wiley and Sons, Inc., 1977.

Webber, Bonnie Lynn. "Logic and Natural Language," Computer, 16: 43-46 (October 1983).

#### Production Rules

Anderson, John R. Language, Memory and Thought. Hillsdale NJ: Lawrence Erlbaum Associates, 1976.

Davis, Randall and Douglas B. Lenat. Knowledge Based Systems in Artificial Intelligence. New York: McGraw-Hill International Book Company, 1982.

Georgeff, M.P. "Procedural Control in Production Systems," Artificial Intelligence, 18: 175-201 (March 1982).

Hayes-Roth, Frederick and others. "Principles of Pattern-Directed Inference Systems," Pattern-Directed Inference Systems, edited by D.A. Waterman and Frederick Hayes-Roth. New York: Academic Press, 1978.

Langley, Pat. "Representational Issues in Learning Systems," Computer, 16: 47-51 (October 1983).

Rosenschein, Stanley J. "The Production System: Architecture and Abstraction," Pattern-Directed Inference Systems, edited by D.A. Waterman and Frederick Hayes-Roth. New York: Academic Press, 1978.

Waterman, D.A. and Frederick Hayes-Roth. "An Overview of Pattern-Directed Inference Systems," Pattern-Directed Inference Systems, edited by D.A. Waterman and Frederick Hayes-Roth. New York: Academic Press, 1978.

Zisman, Michael D. "Use of Production Systems for Modeling Asynchronous Concurrent Processes," Pattern-Directed Inference Systems, edited by D.A. Waterman and Frederick Hayes-Roth. New York: Academic Press, 1978.

### Semantic Networks

Etherington, David W. and Raymond Reiter. "On Inheritance Hierarchies with Exceptions," Proceedings of the Third National Conference on Artificial Intelligence, AAAI-83. 104-108. William Kaufmann, Inc., Los Altos CA, 1983.

Fahlman, Scott E. and others. "Cancellation in a Parallel Semantic Network," Proceedings of the Seventh International Joint Conference on Artificial Intelligence, IJCAI-81, Volume 1. 257-263. William Kaufmann, Inc., Los Altos CA, 1981.

Fox, Mark S. "On Inheritance In Knowledge Representation," Proceedings of the Sixth International Joint Conference on Artificial Intelligence, IJCAI-79, Volume 1. 282-284. William Kaufmann, Inc., Los Altos CA, 1979.

Griffith, Robert L. "Three Principles of Representation for Semantic Networks," ACM Transactions on Database Systems, 7: 417-442 (September 1982).

Laubsch, Joachim H. "Interfacing a Semantic Net with an Augmented Transition Network," Proceedings of the Sixth International Joint Conference on Artificial Intelligence, IJCAI-79, Volume 1. 516-518. William Kaufmann, Inc., Los Altos CA, 1979.

Levesque, Hector and John Mylopoulos. "A Procedural Semantics for Semantic Networks," Associative Networks: The Representation and use of Knowledge by Computers, edited by N.V. Findler. New York: Academic Press, Inc., 1979.

Shapiro, Stuart C. "The SNePS Semantic Network Processing System," Associative Networks: The Representation and use of Knowledge by Computers, edited by N.V. Findler. New York: Academic Press, Inc., 1979.

Woods, William A. "What's in a Link: Foundations for Semantic Networks," Representation and Understanding: Studies in Cognitive Science, edited by Daniel G. Bobrow and Allan Collins. New York: Academic Press, Inc., 1975.

## Frames

Bobrow, Daniel G. and Terry Winograd. "An Overview of KRL, a Knowledge Representation Language," Cognitive Science, 1: 3-46 (1977).

Bobrow, Daniel G. and others. "Experience with KRL-Ø One Cycle of a Knowledge Representation Language," Proceedings of the 5th International Joint Conference on Artificial Intelligence, IJCAI-77, Volume 1. 213-222. William Kaufmann, Inc., Los Altos Ca, 1977.

Kuipers, Benjamin J. "A Frame for Frames: Representing Knowledge for Recognition," Representation and Understanding: Studies in Cognitive Science, edited by Daniel G. Bobrow and Allan Collins. New York: Academic Press, Inc., 1975.

Mylopoulos, John and others. "Building Knowledge-Based Systems: The PSN Experience," Computer, 16: 83-89 (October 1983).

Stefik, Mark. "An Examination of a Frame-Structured Representation System," Proceedings of the Sixth International Joint Conference on Artificial Intelligence, IJCAI-79, Volume 2. 845-852. William Kaufmann, Inc., Los Altos CA, 1979.

Wilks, Yorick. "Making Preferences More Active," Associative Networks: The Representation and use of Knowledge by Computers, edited by N.V. Findler. New York: Academic Press, Inc., 1979.

Winograd, Terry. "Frame Representations and the Declarative/Procedural Controversy," Representation and Understanding: Studies in Cognitive Science, edited by Daniel G. Bobrow and Allan Collins. New York: Academic Press, Inc., 1975.

## Scripts and Conceptual Dependency

Schank, Roger C. and Robert P. Abelson. Scripts, Plans, Goals and Understanding. Hillsdale NJ: Lawrence Erlbaum, 1977.

Schank, Roger C. "The Structure of Episodes in Memory." Representation and Understanding: Studies in Cognitive Science, edited by Daniel G. Bobrow and Allan Collins. New York: Academic Press, Inc., 1975.

### Combined Knowledge Representations

Brachman, Ronald J. and others. "KRYPTON: Integrating Terminology and Assertion." Proceedings of the Third National Conference on Artificial Intelligence, AAAI-83. 31-35. William Kaufmann, Inc., Los Altos CA, 1983.

Brown, John Seely and Richard R. Burton. "Multiple Representations of Knowledge for Tutorial Reasoning," Representation and Understanding: Studies in Cognitive Science, edited by Daniel G. Bobrow and Allan Collins. New York: Academic Press, Inc., 1975.

De Kleer, Johan. "Multiple Representations of Knowledge in a Mechanics Problem-Solver," Proceedings of the 5th International Joint Conference on Artificial Intelligence, IJCAI-77, Volume 1. 299-304. William Kaufmann, Inc., Los Altos CA, 1977.

Goldstein, Ira P. "The Genetic Graph: A Representation for the Evolution of Procedural Knowledge," International Journal of Man-Machine Studies, 11: 51-77 (January 1979).

Janas, Jurgen M. and Camilla B. Schwind. "Extensional Semantic Networks: Their Representation, Application, and Generation," Associative Networks: The Representation and use of Knowledge by Computers, edited by N.V. Findler. New York: Academic Press, Inc., 1979.

### Knowledge Representation Relationships

Borkin, Sheldon A. Data Models: A Semantic Approach for Data Base Systems. Cambridge MA: The MIT Press, 1980.

Deliyanni, Amaryllis and Robert A. Kowalski. "Logic and Semantic Networks," Communications of the ACM, 22: 184-192 (March 1979).

Dilger, Werner and Wolfgang Womann. "Semantic Networks as Abstract Data Types," Proceedings of the Eighth International Joint Conference on Artificial Intelligence, IJCAI-83, Volume 1. 321-324. William Kaufmann, Inc., Los Altos CA, 1983.

Fikes, Richard and Gary Hendrix. "A Network-Based Knowledge Representation and its Natural Deduction System," Proceedings of the 5th International Joint Conference on Artificial Intelligence, IJCAI-77, Volume 1. 235-246. William Kaufmann, Inc., Los Altos CA, 1977.

- Giordana, A. and L. Saitta. "Modeling Production Rules by Means of Predicate Transition Networks," Information Sciences, 35: 1-41 (March 1985).
- Hendrix, Gary G. "Encoding Knowledge in Partitioned Networks," Associative Networks: The Representation and use of Knowledge by Computers, edited by N.V. Findler. New York: Academic Press, Inc., 1979.
- Israel, David J. and Ronald J. Brachman. "Distinctions and Confusions: A Catalogue Raisonne," Proceedings of the Seventh International Joint Conference on Artificial Intelligence, IJCAI-81, Volume 1. 452-459. William Kaufmann, Inc., Los Altos CA, 1981.
- McSkimin, James R. and Jack Minker. "A Predicate Calculus Based Semantic Network for Deductive Searching," Associative Networks: The Representation and use of Knowledge by Computers, edited by N.V. Findler. New York: Academic Press, Inc., 1979.
- . "The Use of a Semantic Network in a Deductive Question-Answering System," Proceedings of the 5th International Joint Conference on Artificial Intelligence, IJCAI-77, Volume 1. 50-58. William Kaufmann, Inc., Los Altos CA, 1977.
- Schubert, L.K. "Extending the Expressive Power of Semantic Networks," Artificial Intelligence, 7: 163-198 (Spring 1976).
- Schubert, Lenhart K. and others. "The Structure and Organization of a Semantic Net for Comprehension and Inference," Associative Networks: The Representation and use of Knowledge by Computers, edited by N.V. Findler. New York: Academic Press, Inc., 1979.
- Simmons, Robert F. and Daniel Chester. "Inferences in Quantified Semantic Networks," Proceedings of the 5th International Joint Conference on Artificial Intelligence, IJCAI-77, Volume 1. 267-273. William Kaufmann, Inc., Los Altos CA, 1977.
- Weiner, James L. and Martha Palmer. "The Design of a System for Designing Knowledge Representation Systems," Proceedings of the Seventh International Joint Conference on Artificial Intelligence, IJCAI-81, Volume 1. 277-282. William Kaufmann, Inc., Los Altos CA, 1981.



## AI and Formal Language Theory Relationships

Abe, Norihiro and others. "Web Grammars and Several Graphs," Journal of Computer and System Sciences, 7: 37-65 (February 1973).

Biermann, Alan W. "The Inference of Regular LISP Programs from Examples," IEEE Transactions on Systems, Man, and Cybernetics, SMC-8: 585-600 (August 1978).

----- . "On the Inference of Turing Machines from Sample Computations," Artificial Intelligence, 3: 181-198 (Fall 1972).

Buneman, O.P. "A Grammar for the Topological Analysis of Plane Figures," Machine Intelligence 5, edited by Bernard Meltzer and Donald Michie. New York: American Elsevier Publishing Company, Inc., 1970.

Clowes, M.B. "Pictorial Relationship--A Syntactic Approach," Machine Intelligence 4, edited by Bernard Meltzer and Donald Michie. New York: American Elsevier Publishing Company, Inc., 1969.

----- . "Perception, Picture Processing and Computers." Machine Intelligence 1, edited by N.L. Collins and Donald Michie. New York: American Elsevier Publishing Company, Inc., 1967.

Chirathamjaree, C. and Martin H. Ackroyd. "A Method for the Inference of Non-Recursive Context-Free Grammars," International Journal of Man-Machine Studies, 12: 379-387 (May 1980).

DeMori, Renato and Lorenza Saitta. "Automatic Learning of Fuzzy Naming Relations Over Finite Languages," Information Sciences, 21: 93-139 (July 1980).

Dietterich, Thomas G. and Ryszard S. Michalski. "Discovering Patterns in Sequences of Events," Artificial Intelligence, 25: 187-232 (February 1985).

Fu, King-Sun, and Taylor L. Booth. "Grammatical Inference: Introduction and Survey--Part II," IEEE Transactions on Systems, Man, and Cybernetics, SMC-5: 409-423 (July 1975).

----- . "Grammatical Inference: Introduction and Survey--Part I," IEEE Transactions on Systems, Man, and Cybernetics, 5: 95-111 (January 1975).

- Hayes-Roth, Frederick. "Uniform Representation of Structural Patterns and an Algorithm for the Induction of Contingency-Response Rules," Information and Control, 33: 87-116 (February 1977).
- Hedrick, Charles L. "Learning a Production System from Examples," Artificial Intelligence, 7: 21-49 (Spring 1976).
- Klette, R. and R. Wiehagen. "Research in the Theory of Inductive Inferences by GDR Mathematicians--A Survey," Information Sciences, 22: 149-169 (November 1980).
- Lin, Wei-Chung and King-Sun Fu. "3D-Plex Grammars," Information Sciences, 34: 1-24 (October 1984).
- Mackworth, Alan K. "How to See a Simple World: An Exegesis of Some Computer Programs for Scene Analysis," Machine Intelligence 8, edited by E.W. Elcock and Donald Michie. Chichester, England: Ellis Horwood Limited, 1977.
- Maurer, H.A. and others. "Using String Languages to Describe Picture Languages," Information and Control, 54: 155-185 (September 1982).
- McLaren, R.W. "A Stochastic Automaton Model for the Synthesis of Learning Systems," IEEE Transactions on Systems Science and Cybernetics, SSC-2: 109-114 (December 1966).
- Milgram, David L. "Web Automata," Information and Control, 29: 162-184 (October 1975).
- Montanari, Ugo G. "Separable Graphs, Planar Graphs and Web Grammars," Information and Control, 16: 243-267 (May 1970).
- Narendra, Kumpati S. and Thathachar, M.A.L. "Learning Automata--A Survey," IEEE Transactions on Systems, Man, and Cybernetics, SMC-4: 323-334 (July 1974).
- Pereira, Fernando C.N. and David H.D. Warren. "Definite Clause Grammars for Language Analysis--A Survey of the Formalism and a Comparison with Augmented Transition Networks," Artificial Intelligence, 13: 231-278 (May 1980).
- Popplestone, R.J. "Freddy in Toyland," Machine Intelligence 4, edited by Bernard Meltzer and Donald Michie. New York: American Elsevier Publishing Company, Inc., 1969.

AD-A172 516

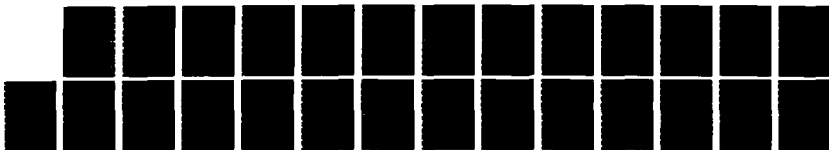
A NON-COGNITIVE FORMAL APPROACH TO KNOWLEDGE  
REPRESENTATION IN ARTIFICIAL (U) AIR FORCE INST OF  
TECH WRIGHT-PATTERSON AFB OH SCHOOL OF ENGI  
J A MCHANNAMA JUN 86 AFIT/DS/ENG/86J-1

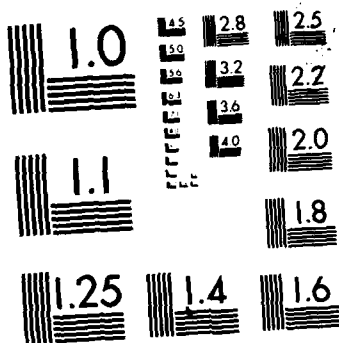
4/4

UNCLASSIFIED

F/G 9/4

NL





MICROCOPY RESOLUTION TEST CHART  
NATIONAL BUREAU OF STANDARDS-1963-A

Rosenfeld, Azriel. "Array Grammar Normal Forms,"  
Information and Control, 23: 173-182 (September 1973).

-----". "Isotonic Grammars, Parallel Grammars and Picture  
Grammars," Machine Intelligence 6, edited by Bernard  
Meltzer and Donald Michie. New York: American Elsevier  
Publishing Company, Inc., 1971.

Siromoney, Gift and others. "Picture Languages with Array  
Rewriting Rules," Information and Control, 22: 447-470  
(June 1973).

Tamura, Shinichi and Kokichi Tanaka. "Learning of Fuzzy  
Formal Language," IEEE Transactions on Systems, Man, and  
Cybernetics, SMC-3: 98-102 (June 1973).

Tsai, Wen-Hsiang and King-Sun Fu. "Attributed Grammar--A  
Tool for Combining Syntactic and Statistical Approaches  
to Pattern Recognition," IEEE Transactions on Systems,  
Man, and Cybernetics, SMC-10: 873-885 (December 1980).

Van der Mude, A. and Adrian Walker. "On the Inference of  
Stochastic Regular Grammars," Information and Control,  
38: 310-329 (September 1978).

Wee, William G. and K.S. Fu. "A Formulation of Fuzzy  
Automata and its Application as a Model of Learning  
Systems," IEEE Transactions on Systems Science and  
Cybernetics, SSC-5: 215-223 (July 1969).

### Appendix C: Invalid Nonenumerable Set

As part of the work in formal language theory, theorists search for examples of sets of words that are not recursively enumerable. These examples provide evidence that Chomsky's hierarchy is a subset of yet a higher set (8:152). However, theorists sometimes propose such sets only to be disproven later. One invalid case was found during work on this dissertation. The case was Manna's construction of a nonrecursively enumerable set (9:39).

For his construction, Manna chose a two letter alphabet,  $Z=\{a,b\}$ . By placing the words of  $Z^*$  in lexicographic order, he could choose the  $j$ th word,  $x_j$ . From a problem in his text, he stated that every Turing machine could be encoded as a word in  $Z^*$ . Therefore, he could choose the  $j$ th Turing machine,  $T_j$ .

With this notation established, he constructed a set  $L_1=\{x_j|x_j \text{ is not accepted by } T_j\}$ . He then stated that  $L_1$  could not be accepted by any Turing machine because the set always contained a word which a given machine would not accept. Therefore,  $L_1$  could not be a recursively enumerable set.

However, he cannot construct  $L_1$  as stated for the set containing every Turing machine. That is, by selecting  $x_j$  from  $Z^*$ , there always exists one machine which accepts every word in  $Z^*$ . Therefore, no  $x_j$  exists for this machine.

This conclusion follows from several definitions given in Section III of this dissertation. From Definition 3.19,  $Z^*$  is a regular set since  $Z$  is a finite, regular set. From Definition 3.21, the  $Z^*$  regular set is accepted by some finite automaton (Turing machine). From Definition 3.18 and the fact that a regular set is a recursive set, this finite automaton accepts every word in the language and rejects all others. But,  $Z^* - Z^* = \emptyset$  and so all words are accepted.

There are several different arguments that could be used in an attempt to counter this disagreement with Manna. It could be argued that  $L_1$  could contain the empty set and then the previously described machine would reject  $L_1$ . However, this would violate the definition of the set  $L_1$ .  $L_1$  contains words, not sets of words. The empty set is a set, not a single word. Language theorists added the empty word to handle this problem. But,  $Z^*$  already contains the empty word by definition.

Another argument possible is that  $L_1$  could be the empty set  $\emptyset$ . Then, the previously described machine would not accept  $L_1$ . This argument fails because  $\emptyset$  is a regular set. By Definition 3.21 of Section III, there exists another machine in the set of all Turing machines that accepts  $\emptyset$ .

Therefore, the set  $L_1$  as constructed by Manna is not a nonrecursively enumerable set.

## Appendix D: Grammar Space

When a set of characteristics for languages are found, the grammars for those languages are usually studied to determine if a related set of characteristics exists for them. As was the case for this dissertation, both the language-space size and grammar-space size are studied. Since the findings of the dissertation were not based on grammar-space size, the main body of the dissertation did not contain the grammar-space size development. Therefore, the results of this related effort are presented in this appendix. In order to remain consistent with the body of the dissertation, the format that was used to present the language-space size effort is also used to present the grammar-space size.

### Supporting Definitions and Theorems

In addition to definitions and theorems already provided in Sections I and III of the main body of this dissertation, an additional set that relate to recursive function theory is needed to be able to handle the generative nature of grammars. In order to provide sufficient information to understand some of these definitions and theorems, related ones are also given. However, only the main definitions and theorems are referred to in the actual proofs.

Also, to remain as consistent as possible with Sections I and III, the same symbolic notation is used where



possible. Therefore,  $Z$  represents a finite alphabet of symbols and  $Z^*$  represents the set of all finite words.

Definition D.01: A partial function,  $f(x_1, x_2, \dots, x_n)$ ,  $n \geq 1$ , mapping  $n$ -tuples of words over  $Z = \{a, b\}$  into words over  $Z$ , is said to be Turing computable if there is a Turing machine,  $M$ , over  $\{a, b, *\}$  which behaves as follows. For every  $n$ -tuple  $(w_1, \dots, w_n)$  of words over  $Z$ ,  $M$  takes the string  $(w_1 * \dots * w_n)$  as input and

1. If  $f(w_1, \dots, w_n)$  is undefined, then  $M$  will loop forever.
2. If  $f(w_1, \dots, w_n)$  is defined, then  $M$  will eventually halt (either accepting or rejecting the input) with a tape containing the value of  $f(w_1, \dots, w_n)$  followed only by blank tape symbols (9:44).

Theorem D.01: A  $n$ -ary partial function mapping  $n$ -tuples of words over  $Z$  into words over  $Z$  is partial recursive if and only if it is Turing computable (9:52).

Definition D.02: A partial recursive function that is defined for all arguments is called a total recursive function (9:52).

Definition D.03: The class of all primitive recursive functions over an alphabet  $Z = \{a, b\}$  is defined as follows (9:45):

1. Base functions:

$\text{nil}(x) = e$  (empty word)

$\text{consa}(x) = ax$

$\text{consb}(x) = bx$

2. Composition:

If  $h, g_1, \dots, g_m$  are primitive recursive then so is the  $n$ -ary function  $f(x_1, \dots, x_n) = h[g_1(x_1, \dots, x_n), \dots, g_m(x_1, \dots, x_n)]$ .

3. Primitive recursion:

If  $g, h_1$  and  $h_2$  are primitive recursive (some arguments of  $h_1$  and  $h_2$  may be missing) then so is the  $n$ -ary function ( $e$  represents the empty word):

a. For  $n=1$ :

$f(e) = w, w \in Z^*$

$f(ax) = h_1[x, f(x)]$

$f(bx) = h_2[x, f(x)]$

b. For  $n \geq 2$ :

$f(e, x_2, \dots, x_n) = g(x_2, \dots, x_n)$ .

$f(ax_1, x_2, \dots, x_n) =$

$h_1[x_1, \dots, x_n, f(x_1, \dots, x_n)]$ .

$f(bx_1, x_2, \dots, x_n) =$

$h_2[x_1, \dots, x_n, f(x_1, \dots, x_n)]$ .

Theorem D.02: The following hierarchy exist for functions  
(9:52; 92:Chapter 13):

All Functions  
     $\cup$   
Partial Recursive Functions  
     $\cup$   
Total Recursive Functions  
     $\cup$   
Primitive Recursive Functions

Now that the recursive functions have been identified, the tie between languages and these functions needs to be specified.

Definition D.04: A language L is Turing recognizable if it is the domain of a Turing realizable function (92:460).

Theorem D.03: A set is Turing recognizable if and only if it is Turing enumerable (92:489).

A final theorem is needed which forms the starting seed for the theorems that follow.

Theorem D.04: The set of base functions of the primitive recursive functions is countably infinite (92:525).

#### Grammar-Space Size

To determine the size of grammar space, the size of the recursive function space needs to be determined first.

Theorem D.05: The cardinality of the set of partial recursive functions is countably infinite.

Proof: First, from Theorem D.04, a subset of the primitive recursive functions has countably

infinite cardinality. From Theorem D.02, this subset is contained within the partial recursive functions. Therefore, the set of partial recursive functions must be at least countably infinite.

Next, Definitions 3.22 and 3.23 of Section III and Theorem D.01 establishes that Turing machines can only compute partial recursive functions. Theorem 3.13 of Section III established that the cardinality of the set of all Turing machines is countably infinite. Therefore, the set of partial recursive functions can be at most countably infinite.

Finally, given that both the upper and lower cardinality-bounds are equal, the set of partial recursive functions is countably infinite. Q.E.D.

Having established the size of computable functions, the relationship between grammars and partial recursive functions is needed. Even though this relationship is well known (92:Chapter 13), the statement of the theorem and the proof technique used herein provides a clear view of this relationship.

Theorem D.06: For every set of type-0 grammars generating the same type-0 language over a given alphabet, there must exist a set of partial recur-

sive functions that generate an equivalent recursively enumerable set (type-0 language).

Proof: The case for one grammar and one partial recursive function will be handled first. From Definitions 3.17 and 3.20 and Theorem 3.12 of Section III, every type-0 language is generated by some type-0 grammar and the language is a recursively enumerable set. Next, from Definition 3.17 and Theorem D.03, every one of the unique type-0 languages is Turing enumerable. Now, by Definition 3.24 of Section III, there must exist a Turing computable function that generates an equivalent, unique, recursively enumerable set of the natural numbers. Finally, by Theorem D.01, the Turing computable function performing the enumeration must be a partial recursive function. Therefore, for some type-0 grammar there must exist some partial recursive function that generates an equivalent recursively enumerable set.

Now, expand this finding to the set of grammars and functions. Because of the existence of normal-forms for the type-0 grammars (6:Chapter 5), more than one type-0 grammar can exist that generates the same type-0 language. Similarly, more than one

partial recursive function exists that generates the same equivalent recursively enumerable set (e.g.  $(4X+4)/2$  and  $2X+2$ ). Therefore, from the previous proof of the single language case, it follows that for every set of type-0 grammars generating the same type-0 language, there must exist a set of partial recursive functions that generate an equivalent recursively enumerable set. Q.E.D.

Theorem D.06 only establishes that some set of partial recursive functions exists for a given set of type-0 grammars. The theorem does not indicate a one-to-one correspondence of the members of these sets. However, a one-to-one correspondence has to be assumed to exist in order to bound the size of grammar space.

Theorem D.07: The cardinality of the set of all type-0 grammars over a given alphabet is countably infinite.

Proof: Because of the existence of normal-forms for the type-0 grammars (6:Chapter 5), more than one type-0 grammar can exist that generates the same type-0 language. However, formal language theorists have not proven the exact number of equivalent type-0 grammars existing per type-0 language. Therefore, the size of grammar space will have to be determined by

establishing upper and lower size-bounds for the set of all type-0 grammars.

The lower size-bound can be established from Definitions 3.07 and 3.20 and Theorem 3.11 of Section III. These definitions and theorem establish that there exists a countably infinite number of unique type-0 languages each generated by a different type-0 grammar. Since a countably infinite number of type-0 languages exist, there must be at least a countably infinite number of unique type-0 grammars. Hence, the lower size-bound on type-0 grammars is countably infinite.

The upper size-bound follows directly from Theorems D.05 and D.06. Theorem D.06 established the existence of a set of type-0 grammars and a set of partial recursive functions that generate an equivalent recursively enumerable set (type-0 language). Because a single type-0 grammar (partial recursive function) only generates a single type-0 language (recursively enumerable set), all existing sets of type-0 grammars (partial recursive functions) meeting Theorem D.06 have to be disjoint. That is, individual type-0 grammar  $G_1$  (partial recursive function  $F_1$ ) can only be a member of one set of type-0 grammars

(partial recursive functions). From this disjoint property, set theory reveals that the largest possible number of sets of type-0 grammars (partial recursive functions) that can meet Theorem D.06 are sets that contain only one member. That is, type-0 grammar  $G_1$  and partial recursive  $F_1$  are the only grammar and function generating the same equivalent recursively enumerable set (type-0 language).

Therefore, to establish the upper size-bound on the set of type-0 grammars, assume that this one-to-one correspondence exists. Then, from Theorem D.05, the upper size-bound on the type-0 grammars is countably infinite.

Finally, since both the upper and lower size-bounds are countably infinite, the cardinality of the set of type-0 grammars is countably infinite. Q.E.D.

Now that the size of the set of type-0 grammars for a given alphabet has been established, the cardinality of the set of all type-0 grammars over all finite alphabets can be found using set theory.

Theorem D.09: The set of type-0 grammars over all finite alphabets has countably infinite cardinality.

Proof: From AFL theory, all the finite alphabets are obtained from a countably infinite alphabet (97:33). From Theorem 3.10 of Section III,



the set of all possible finite alphabets over this countably infinite alphabet has countably infinite cardinality. Since every set has the empty set  $\emptyset$  as a finite subset, this countably infinite group of finite sets must contain  $\emptyset$ . However, by Definition 3.01 of Section III, an alphabet cannot be the empty set. Therefore, remove the empty set from this group of alphabets. By applying the  $f(x)=x-1$  mapping on the indexes of the remaining finite sets, there is still a countably infinite number of finite sets remaining.

Now, for each finite alphabet, form the countably infinite set of all type-0 grammars (see Theorem D.07). Then, perform the set union operation over these countably infinite sets of grammars. By Theorem 3.05 of Section III, the resulting set has countably infinite cardinality. Q.E.D.

## Bibliography

1. Cohen, Paul R. and Edward A. Feigenbaum, editors. The Handbook of Artificial Intelligence, Volume 3. Los Altos CA: William Kaufmann, Inc., 1982.
2. Sanderson, Authur C. and Yeheshua Y. Zeevi, editors. "Special Issue on Neural and Sensory Information Processing," IEEE Transactions on Systems, Man, and Cybernetics, SMC-13: (September/October 1983).
3. Horowitz, Ellis and Sartaj Shani. Fundamentals of Data Structures. Rockville MD: Computer Science Press, Inc., 1983.
4. Rattner, Justin and George Cox. "Object-Based Computer Architecture," Computer Architecture News, 8: 4-11 (October 1980).
5. Barr, Avron and Edward A. Feigenbaum, editors. The Handbook of Artificial Intelligence, Volume 1. Los Altos CA: William Kaufmann Inc., 1981.
6. Revesz, Gyorgy E. Introduction to Formal Languages. New York: McGraw-Hill Book Company, 1983.
7. Ginsburg, Seymour. Algebraic and Automata-Theoretic Properties of Formal Languages. New York: American Elsevier Publishing Co., 1975.
8. Chomsky, Noam. "On Certain Formal Properties of Grammars," Information and Control, 2: 137-167 (June 1959).
9. Manna, Zohar. Mathematical Theory of Computation. New York: McGraw-Hill Book Company, 1974.
10. Rosenberg, Arnold L. "Real-Time Definable Languages," Journal of the ACM, 14: 645-662 (October 1967).
11. Ginsburg, Seymour and Michael A. Harrison. "One-Way Nondeterministic Real-Time List-Storage Languages," Journal of the ACM, 15: 429-446 (July 1968).
12. Book, Ronald V. "Time-Bounded Grammars and Their Languages," Journal of Computer and System Sciences, 5: 397-429 (August 1971).

13. Book, Ronald V. "Tally Languages and Complexity Classes," Information and Control, 26: 186-193 (October 1974).
14. Fischer, Patrick C. and others. "Counter Machines and Counter Languages," Mathematical Systems Theory, 2: 265-283 (1968).
15. Walljasper, S.J. "Left-Derivation Bounded Languages," Journal of Computer and System Sciences, 8: 1-7 (February 1974).
16. Kriegel, H.P. and H.A. Maurer. "Formal Translations and Szilard Languages," Information and Control, 30: 187-198 (February 1976).
17. Rosenkrantz, Daniel J. "Programmed Grammars and Classes of Formal Languages," Journal of the ACM, 16: 107-131 (January 1969).
18. Greibach, Sheila and John Hopcroft. "Scattered Context Grammars," Journal of Computer and System Sciences, 3: 233-247 (August 1969).
19. Mayoh, Brian H. "Attribute Grammars and Mathematical Semantics," SIAM Journal on Computing, 10: 503-518 (August 1981).
20. Aho, Alfred V. "Indexed Grammars--An Extension of Context Free Grammars," Journal of the ACM, 15: 647-671 (October 1968).
21. Gabrielian, Armen and Seymour Ginsburg. "Grammar Schemata," Journal of the ACM, 21: 213-226 (April 1974).
22. Abraham, Samuel. "Compound and Serial Grammars," Information and Control, 20: 432-438 (June 1972).
23. Salomaa, Arto. "Grammatical Families," Lecture Notes in Computer Science, Number 85, edited by J.W. deBakker and J. van Leeuwen. Berlin: Springer-Verlag, 1980.
24. Ginsburg, Seymour and Sheila Greibach. "Abstract Families of Languages," Memoirs of the American Mathematical Society, Number 87, edited by Seymour Ginsburg and others. Providence RI: American Mathematical Society, 1969.
25. Greibach, Sheila A. "Full AFLs and Nested Iteration Substitution," Information and Control, 16: 7-35 (March 1970).

26. Ginsburg, Seymour and others. "Pre-AFL," Memoirs of the American Mathematical Society, Number 87, edited by Seymour Ginsburg and others. Providence RI: American Mathematical Society, 1969.
27. Ginsburg, Seymour and Sheila Greibach. "Principal AFL," Journal of Computer and System Sciences, 4: 308-338 (August 1970).
28. Paul, W.J. "On Time Hierarchies," Journal of Computer and System Sciences, 19: 197-202 (October 1979).
29. Aho, A.V. and J.D. Ullman. "Syntax Directed Translations and the Pushdown Assembler," Journal of Computer and System Sciences, 3: 37-56 (February 1969).
30. Kasai, Takumi. "A Hierarchy Between Context-Free and Context-Sensitive Languages," Journal of Computer and System Sciences, 4: 492-508 (October 1970).
31. Engelfriet, Joost. "Hierarchies of Hyper-AFLs," Journal of Computer and System Sciences, 30: 86-115 (February 1985).
32. Hermann, K. and G. Walter. "Structural Equivalence of Context-Free Grammar Forms is Decidable," Lecture Notes in Computer Science, Number 52, edited by Arto Salomaa and Magnus Steinby. Berlin: Springer-Verlag, 1977.
33. Schnorr, C.P. "Transformational Classes of Grammars," Information and Control, 14: 252-277 (March 1969).
34. Senizergues, Geraud. "The Equivalence Problem for N.T.S. Languages is Decidable," Lecture Notes in Computer Science, Number 145. Berlin: Springer-Verlag, 1982.
35. McNaughton, Robert. "Parenthesis Grammars," Journal of the ACM, 14: 490-500 (July 1967).
36. Herman, Gabor T. "Closure Properties of Some Families of Languages Associated with Biological Systems," Information and Control, 24: 101-121 (February 1974).
37. Ehrig, Hartmut and others, editors. Lecture Notes in Computer Science, Number 153. Berlin: Springer-Verlag, 1982.
38. Blikle, Andrzej. "Equational Languages," Information and Control, 21: 134-147 (September 1972).

39. Ezawa, Yoshinori and others. "Interactive Languages," Journal of Computer and System Sciences, 12: 49-63 (February 1976).
40. Vere, Steven A. "Relational Production Systems," Artificial Intelligence 8: 47-68 (February 1977).
41. Witteveen, Cees and Harrie Boelens. "Inferring Control Structures from the Behavior of a Production System," Information and Control, 51: 275-301 (December 1981).
42. Mitchell, Tom and others, editors. "Special Issue on Machine Learning," ACM SIGART Newsletter, Number 76, (April 1981).
43. Osherson, Daniel N. and others. "Ideal Learning Machines," Cognitive Science, 6: 277-290 (July-September 1982).
44. Graham, James H. and George N. Saridis. "Linguistic Decision Structures for Hierarchical Systems," IEEE Transactions on Systems, Man, and Cybernetics, SMC-12: 325-333 (May/June 1982).
45. Pfaltz, John L. and Azriel Rosenfeld. "Web Grammars," Proceedings of the International Joint Conference on Artificial Intelligence, IJCAI-69. 609-619. William Kaufmann, Inc., Los Altos CA, 1969.
46. You, Kai Ching and King-Sun Fu. "A Syntactic Approach to Shape Recognition Using Attributed Grammars," IEEE Transactions on Systems, Man, and Cybernetics, SMC-9: 334-345 (June 1979).
47. Feder, Jerome. "Plex Languages," Information Sciences, 3: 225-241 (1971).
48. Church, Alonzo. Introduction to Mathematical Logic I. Princeton NJ: Princeton University Press, 1956.
49. Robinson, J.A. "A Machine-Oriented Logic Based on the Resolution Principle," Journal of the ACM, 12: 23-41 (January 1965).
50. Post, Emil L. "Formal Reductions of the General Combinatorial Decision Problem," American Journal of Mathematics, 65: 197-268 (April 1943).
51. Newell, A. and H.A. Simon. Human Problem Solving. Englewood Cliffs NJ: Prentice-Hall, 1972.

52. Quillian, M. Ross. "Semantic Memory," Semantic Information Processing, edited by Marvin Minsky. Cambridge MA: MIT Press, 1968.
53. Brachman, Ronald J. "On the Epistemological Status of Semantic Networks," Associative Networks: The Representation and use of Knowledge by Computers, edited by N.V. Findler. New York: Academic Press, Inc., 1979.
54. Brachman, Ronald J. "What IS-A is and Isn't: An Analysis of Taxonomic Links in Semantic Networks," Computer, 16: 30-36 (October 1983).
55. Minsky, Marvin. "A Framework for Representing Knowledge," The Psychology of Computer Vision, edited by Patrick Henry Winston. New York: McGraw-Hill, 1975.
56. Winston, Patrick Henry. Artificial Intelligence. Reading MA: Addison-Wesley Publishing Company, 1979.
57. Schank, Roger C. and Robert P. Abelson. "Scripts, Plans, and Knowledge," Advance Papers from the Fourth International Joint Conference on Artificial Intelligence, IJCAI-75. 151-157. William Kaufmann, Inc., Los Altos CA, 1975.
58. Schank, Roger C. and Lawrence G. Tesler. "A Conceptual Parser for Natural Language," Proceedings of the International Joint Conference on Artificial Intelligence, IJCAI-69. 569-578. William Kaufmann, Inc., Los Altos CA, 1969.
59. Schank, Roger C. "Identification of Conceptualizations Underlying Natural Language," Computer Models of Thought and Language, edited by R.C. Schank and K.M. Colby. San Francisco: W.H. Freedman and Company, 1973.
60. Schank, Roger. "Representation and Understanding of Text," Machine Intelligence 8, edited by E.W. Elcock and Donald Michie. Chichester, England: Ellis Horwood Limited, 1977.
61. Rich, Elaine. Artificial Intelligence. New York: McGraw-Hill Book Company, 1983.
62. Hayes, Philip J. "On Semantic Nets, Frames and Associations," Proceedings of the 5th International Joint Conference on Artificial Intelligence--1977, IJCAI-77, Volume 1. 99-107. William Kaufmann, Inc., Los Altos CA, 1977.

63. Brachman, Ronald J. and others. "Krypton: A Functional Approach to Knowledge Representation," Computer, 16: 67-73 (October 1983).
64. Charniak, Eugene. "A Common Representation for Problem-Solving and Language-Comprehension Information," Artificial Intelligence, 16: 225-255 (July 1981).
65. Laubsch, Joachim H. "Some Thoughts About Representing Knowledge in Instructional Systems," Advance Papers from the Fourth International Joint Conference on Artificial Intelligence, IJCAI-75. 122-125. William Kaufmann, Inc., Los Altos CA, 1975.
66. Georgeff, Michael and Umberto Bonollo. "Procedural Expert Systems," Proceedings of the Eighth International Joint Conference on Artificial Intelligence, IJCAI-83, Volume 1. 151-157. William Kaufmann, Inc., Los Altos CA, 1983.
67. Hayes, P.J. "In Defence of Logic," Proceedings of the 5th International Joint Conference on Artificial Intelligence--1977, IJCAI-77, Volume 1. 559-565. William Kaufmann, Inc., Los Altos CA, 1977.
68. Dilger, Werner and Wolfgang Womann. "The METANET: A Means for the Specification of Semantic Networks as Abstract Data Types," International Journal of Man-Machine Studies, 21: 463-492 (December 1984).
69. Hayes P.J. "The Logic of Frames," Readings in Artificial Intelligence, edited by Bonnie Lynn Webber and Nils J. Nilsson. Palo Alto CA: Tioga Publishing Company, 1981.
70. Simmons, Robert J. and Bertram C. Bruce. "Some Relations Between Predicate Calculus and Semantic Net Representations of Discourse," Advance Papers from the Second International Joint Conference on Artificial Intelligence, IJCAI-71. 524-530. William Kaufmann, Inc., Los Altos Ca, 1971.
71. Sandewall, Erik. "A Set-Oriented Property-Structure Representation for Binary Relations SPB," Machine Intelligence 5, edited by Bernard Meltzer and Donald Michie. New York: American Elsevier Publishing Company, Inc., 1970.

72. Hendrix, Gary G. "Expanding the Utility of the Semantic Networks Through Partitioning," Advance Papers from the Fourth International Joint Conference on Artificial Intelligence, IJCAI-75. 115-121. William Kaufmann, Inc., Los Altos CA, 1975.
73. Duda, Richard O. and others. "Semantic Network Representations in Rule-Based Inference Systems," Pattern-Directed Inference Systems, edited by D.A. Waterman and Frederick Hayes-Roth. New York: Academic Press, 1978.
74. Rychener, Michael D. "A Semantic Network of Production Rules in a System for Describing Computer Structures," Proceedings of the Sixth International Conference on Artificial Intelligence, IJCAI-79, Volume 2. 738-743. William Kaufmann, Inc., Los Altos CA, 1979.
75. Bobrow, Daniel G. "Dimensions of Representation," Representation and Understanding: Studies in Cognitive Science, edited by Daniel G. Bobrow and Allan Collins. New York: Academic Press, Inc., 1975.
76. Newell, Allen. "The Knowledge Level," Artificial Intelligence, 18: 87-127 (January 1982).
77. Reimer, Ulrich and Udo Hahn. "A Formal Approach to the Semantics of a Frame Data Model," Proceedings of the Eighth International Joint Conference on Artificial Intelligence, IJCAI-83, Volume 1. 337-339. William Kaufmann, Inc., Los Altos CA, 1983.
78. Schubert, L.K. "Extending the Expressive Power of Semantic Networks," Advance Papers from the Fourth International Joint Conference on Artificial Intelligence, IJCAI-75. 158-164. William Kaufmann, Inc., Los Altos CA, 1975.
79. Brachman, Ronald J. "What's in a Concept: Structural Foundations for Semantic Networks," International Journal of Man-Machine Studies, 9: 127-152 (March 1977).
80. Davis, Randall and Jonathan King. "An Overview of Production Systems," Machine Intelligence 8, edited by E.W. Elcock and Donald Michie. Chichester, England: Ellis Horwood Limited, 1977.
81. Levesque, Hector J. "Foundations of a Functional Approach to Knowledge Representation," Artificial Intelligence, 23: 155-212 (July 1984).



82. McDermott, J. and others. "The Efficiency of Certain Production System Implementations," Pattern-Directed Inference Systems, edited by D.A. Waterman and Frederick Hayes-Roth. New York: Academic Press, 1978.
83. Orlowska, Ewa and Zdzislaw Pawlak. "Expressive Power of Knowledge Representation Systems," International Journal of Man-Machine Studies, 20: 485-500 (May 1984).
84. Brachman, R.J. and B.C. Smith, editors. "Special Issue on Knowledge Representation," ACM SIGART Newsletter, Number 70: (February 1980).
85. Feder, Jerome. "Languages of Encoded Line Patterns," Information and Control, 13: 230-244 (September 1968).
86. Hamblin, C.L. "Language Types and Logical Theorems," Information and Control, 22: 183-187 (March 1973).
87. Smith, Brian Cantwell. Levels, Layers, and Planes: The Framework of a System of Knowledge Representation Semantics. MS Thesis. Artificial Intelligence Laboratory, Massachusetts Institute of Technology, Cambridge, MA, February 1978.
88. Bertoni, A. and others. "Equivalence and Membership Problems for Regular Trace Languages," Lecture Notes in Computer Science, Number 140. Berlin: Springer-Verlag, 1982.
89. Hofstadter, Douglas R. Godel, Escher, Bach: An Eternal Golden Braid (Vintage Books Edition). New York: Random House, Inc., 1980.
90. Stanat, Donald F. and David F. McAllister. Discrete Mathematics in Computer Science. Englewood Cliffs NJ: Prentice-Hall, Inc., 1977.
91. Rotman, B. and G.T. Kneebone. The Theory of Sets and Transfinite Numbers. New York: American Elsevier Publishing Company, Inc., 1968.
92. Denning, Peter J. and others. Machines, Languages, and Computation. Englewood Cliffs NJ: Prentice-Hall, Inc., 1978.
93. Kain, Richard Y. Automata Theory: Machines and Languages. New York: McGraw-Hill Book Company, 1972.
94. Rayward-Smith, V.J. A First Course in Formal Language Theory. Oxford Great Britain: Blackwell Scientific Publications, 1983.

95. Ginsburg, Seymour. The Mathematical Theory of Context Free Languages. New York: McGraw-Hill Book Company, 1966.
96. Harrison, Michael A. Introduction to Formal Language Theory. Reading MA: Addison-Wesley Publishing Company, 1978.
97. Greibach, Sheila and John Hopcroft. "Independence of AFL Operations," Memoirs of the American Mathematical Society, Number 87, edited by Seymour Ginsburg and others. Providence RI: American Mathematical Society, 1969.
98. Haines, Leonard Harold. Generation and Recognition of Formal Languages. PhD dissertation. Mathematics. Massachusetts Institute of Technology, Cambridge MA, June 1965.
99. Maurer, H.A. and others. "On Finite Grammar Forms," International Journal of Computer Mathematics, 12: 227-240 (February 1983)
100. Maurer, H.A. and others. "On Predecessors of Finite Languages," Information and Control, 50: 259-275 (September 1981).
101. Hayes-Roth, Frederick and others, editors. Building Expert Systems. Reading MA: Addison-Wesley Publishing Company, 1983.
102. Partsch, H. and R. Steinbruggen. "Program Transformation Systems," ACM Computing Surveys, 15: 199-236 (September 1983).
103. Conrad, Michael. "On Design Principles for a Molecular Computer," Communications of the ACM, 28: 464-480 (May 1985).
104. Routh, Maj R. L. Cortical Thought Theory-A Working Model of the Human Gestalt Mechanism. PhD dissertation. School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, July 1985 (AD-A163 215).
105. Aho, Alfred V. and others. The Design and Analysis of Computer Algorithms. Reading MA: Addison-Wesley Publishing Company, 1974.

### Vita

Major Jim A. McMannama was born 29 April 1947 in Larned, Kansas and graduated from Larned High School in 1965. Subsequently, he attended Wichita State University from which he received the degrees of Bachelor of Science (June 1970) and Master of Science (June 1972) in Electrical Engineering. He also received a commission in the USAF through the ROTC program. He began active duty in July 1972 and served as a digital avionics engineer for the F-111 aircraft at the Sacramento Air Logistics Center, McClellan AFB, California, until October 1976. Then, he served as a computer and integration engineer for the development of the B-52 Offensive Avionics System (OAS) at the Aeronautical Systems Division, Wright-Patterson AFB, Ohio, until October 1979. Then, he served as the lead software test engineer for the OAS test program at Edwards AFB's Combined Test Force, Boeing Military Airplane Company, Wichita, Kansas, until December 1981. Then, he served as engineering project manager for the B-1B computer systems at the Aeronautical Systems Division, Wright-Patterson AFB, Ohio, until entering the School of Engineering, Air Force Institute of Technology, in July 1983.

Permanent address: 924 State Street

Larned, Kansas 67550

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE

A1172.5/6

## REPORT DOCUMENTATION PAGE

1. REPORT SECURITY CLASSIFICATION <b>UNCLASSIFIED</b>		1b. RESTRICTIVE MARKINGS	
2a. SECURITY CLASSIFICATION AUTHORITY		3. DISTRIBUTION/AVAILABILITY OF REPORT <b>Approved for public release; distribution unlimited.</b>	
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE		5. MONITORING ORGANIZATION REPORT NUMBER(S)	
4. PERFORMING ORGANIZATION REPORT NUMBER(S) <b>AFIT/DS/ENG/86J-1</b>		7a. NAME OF MONITORING ORGANIZATION	
6a. NAME OF PERFORMING ORGANIZATION <b>School of Engineering</b>	6b. OFFICE SYMBOL (If applicable) <b>AFIT/ENG</b>	7b. ADDRESS (City, State and ZIP Code)	
6c. ADDRESS (City, State and ZIP Code) <b>Air Force Institute of Technology Wright-Patterson AFB, Ohio 45433</b>		9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER	
8a. NAME OF FUNDING/SPONSORING ORGANIZATION	8b. OFFICE SYMBOL (If applicable)	10. SOURCE OF FUNDING NOS.	
8c. ADDRESS (City, State and ZIP Code)		PROGRAM ELEMENT NO.	TASK NO. 1AW AFR 190-17
11. TITLE (Include Security Classification) <b>See Box 19</b>		WORK UNIT NO. <b>3544</b>	
12. PERSONAL AUTHOR(S) <b>Jim A. McMannama, B.S... M.S... Major, USAF</b>			
13a. TYPE OF REPORT <b>PhD Dissertation</b>	13b. TIME COVERED FROM _____ TO _____	14. DATE OF REPORT (Yr., Mo., Day) <b>1986 June</b>	15. PAGE COUNT <b>312</b>
16. SUPPLEMENTARY NOTATION			
17. COSATI CODES		18. SUBJECT TERMS (Continue on reverse if necessary)	
FIELD <b>09</b>	GROUP <b>02</b>	SUB. GR. <b>Artificial Intelligence, Syntax, Machine Translation, Grammars, Linguistics, Programming Languages, Expert Systems</b>	
19. ABSTRACT (Continue on reverse if necessary and identify by block number)			
11. Title: A NON-COGNITIVE FORMAL APPROACH TO KNOWLEDGE REPRESENTATION IN ARTIFICIAL INTELLIGENCE			
Chairman of the Advisory Committee: Gary B. Lamont Professor of Electrical Engineering			
20. DISTRIBUTION/AVAILABILITY OF ABSTRACT <b>UNCLASSIFIED/UNLIMITED XX SAME AS RPT. <input type="checkbox"/> DTIC USERS <input type="checkbox"/></b>		21. ABSTRACT SECURITY CLASSIFICATION <b>UNCLASSIFIED</b>	
22a. NAME OF RESPONSIBLE INDIVIDUAL <b>Gary B. Lamont, Professor</b>		22b. TELEPHONE NUMBER (Include Area Code) <b>513-255-3576</b>	22c. OFFICE SYMBOL <b>AFIT/ENG</b>

With the entry of Artificial Intelligence (AI) into real-time applications, a rigorous analysis of AI expert systems is required in order to validate them for operational use. To satisfy this requirement for analysis of the associated knowledge representations, the techniques of formal language theory are used. A combination of theorems, proofs and problem-solving techniques from formal language theory are employed to analyze language equivalents of the more commonly used AI knowledge representations of production rules (excluding working memory or situation data) and semantic networks.

Using formal language characteristics, it is shown no single support-tool or automatic-programming tool can ever be constructed that can handle all possible production-rule or semantic-network variations. Also, it is shown that the entire set of finite production-rule languages is able to be stored in and retrieved from finite semantic-network languages. In effect, the semantic-network structure is shown to be a viable candidate for a centralized database of knowledge.

END

11-86

DTIC